Virtual therapist for aphasia treatment

submitted by
Annamaria Pompili

Thesis Supervisor: Prof. Dr. M. T. Pazienza, Tor Vergata Rome
Thesis Advisor: Prof. Dr. I. M. Trancoso, INESC-ID Lisboa / IST Lisboa
Thesis Advisor: Dr. A. Abad, INESC-ID Lisboa

Knowledge-Based Systems
Rome University of Tor Vergata, Department of Computer Science, Systems and Production
Via del Politecnico 1 - 00133 Roma

in cooperation with

L²F – Spoken Language Systems Laboratory
INESC ID-Lisboa, Rua Alves Redol 9, 1000-029 Lisboa, Portugal

Lisbon/Rome, April 2011
Sommario

L’afasia è un disturbo del linguaggio che si verifica in seguito a lesioni cerebrali. Può influenzare la produzione del linguaggio, la comprensione orale, oltre che la capacità di leggere e scrivere correttamente. Questo tipo di disturbo del linguaggio comporta innumerevoli problemi professionali, familiari ed economici, sia da un punto di vista dell’individuo che della società. Diversi studi sull’afasia hanno dimostrato l’effettiva efficacia di terapie basate su sessioni di esercizi di logopedia ed evidenziato, d’altro canto, quanto sia importante l’assiduità nella pratica delle sessioni terapeutiche al fine di garantire una sostanziale e rapida riabilitazione. Purtroppo, però, ciò non è sempre possibile o agevole per il paziente, a causa di numerose ragioni pratiche, come l’impossibilità fisica di raggiungere il luogo dove si svolgono le sedute di terapia o per mancanza di tempo.

Questa tesi rappresenta la prima pietra miliare nel processo di costruzione di un sofisticato terapeuta virtuale per il trattamento dell’afasia (VITHEA). Il sistema è stato implementato come una applicazione web, così da risultare altamente portabile e consentire di svolgere le sessioni terapeutiche ovunque ed in qualsiasi momento. Oltre a consentire ai terapeuti la creazione di nuovi esercizi attraverso una interfaccia intuitiva, web based, Vithea permette anche di monitorare le prestazioni dei pazienti, sia in termini di frequenza di accessi al sistema che dei progressi ottenuti.

Il terapeuta virtuale è stato realizzato integrando diverse tecnologie di natura eterogenea. La parte utente, infatti, sfrutta tecnologia Flash® per supportare una ricca interazione multimediale (ad esempio la memorizzazione e l’ascolto di audio) all’interno di un convenzionale browser Web. Lato server, l’applicazione è strutturata utilizzando una serie di avanzati frameworks open source per lo sviluppo di applicazioni Web, così da garantire un’elevata estensibilità e manutenibilità del sistema. Per quanto concerne la valutazione automatica della correttezza delle espressioni vocali dei pazienti, Vithea sfrutta un avanzato sistema di riconoscimento vocale automatico, AUDIMUS, sviluppato dal gruppo L²F dell’INESC-ID di Lisbona in oltre vent’anni di ricerca nel settore.
Abstract

Aphasia is a language disorder that occurs after brain injuries. It can affect speech production, hearing comprehension, reading and writing. This kind of language disorder entails countless social, professional, familiar and financial problems, both from a point of view of the individual and the society. Several studies about aphasia have demonstrated the positive effects of speech therapy activities. In particular, the more intensive and frequent are the therapy sessions, the more positive and quicker is the rehabilitation. Unfortunately, this is not always possible due to several practical reasons like for instance the impossibility to physically reach the place where therapies are carried out or time constraints.

This thesis represents the first milestone in the process of building a sophisticated Virtual Therapist for Aphasia Treatment (VITHEA). The system has been built as a Web Application, thus it is highly portable and allows to carry out the therapy sessions everywhere and at any time. Beside the possibility to design new therapeutic exercises via an intuitive, web based, What-You-See-Is-What-You-Get interface, Vithea provides also the functionality to monitor patient’s performance in terms of frequency of access to the system and therapeutical progress.

The virtual therapist has been achieved by integrating a number of technologies. The client side relies on a Flash\textsuperscript{\textregistered} component which allows to support a rich multi-media interaction for recording and playing audio. On the other hand, the server side relies on mainstream Web Application Frameworks to ensure extensibility and maintainability and on Automatic Speech Recognition in order to evaluate patient’s training sessions.
Resumo

A afasia é uma perturbação na função da linguagem resultante de danos cerebrais. Pode afectar a produção do discurso, a compreensão auditiva, a capacidade de leitura e de escrita. Esta deteriorização do uso da linguagem representa inúmeros problemas a nível social, profissional, familiar e económico, tanto do ponto de vista do indivíduo como da sociedade. Vários estudos sobre a afasia têm evidenciado os efeitos positivos de actividades de terapia da fala, demonstrando que quanto mais frequentes e intensivas forem as sessões, mais rápida será a reabilitação. Infelizmente, estes tratamentos nem sempre são possíveis de realizar devido a razões práticas, como a impossibilidade física de deslocação ao sítio onde a terapia terá lugar ou mesmo questões de tempo.

Esta tese representa uma primeira abordagem ao processo de construção de um sofisticado Terapeuta Virtual para o Tratamento da Afasia (VITHEA). O sistema foi construído como uma Aplicação Web, assim é altamente portátil e permite que a terapia se desenvolva em qualquer lugar, a qualquer hora. Além de permitir a criação de novos exercícios terapêuticos através de uma interface intuitiva, What-You-See-Is-What-You-Get, Vithea possui também a funcionalidade de monitorizar o desempenho do paciente, tendo em consideração a frequência dos acessos aos sistema e o progresso do paciente.

O terapeuta virtual foi concebido através da integração de várias tecnologias. A parte do utilizador é realizada em Flash®, permitindo uma rica interacção multimedial para gravar e tocar ficheiros audio. Por outro lado, a parte do servidor é suportada por Frameworks para aplicações Web que garantem a extensibilidade e manutenção e, por último, o Reconhecimento Automático da Fala que consente avaliar as sessões dos pacientes.
Acknowledgments

I would like to take this opportunity to express my gratitude to the many people whose support made this thesis possible.

I am heartily thankful to my supervisor, Prof. Dr. Maria Teresa Pazienza, for her constant availability and to have provided me with the opportunity to develop my thesis in the context of an exciting international research collaboration. Without her support, this work would have never been possible.

Further, my gratitude goes to Dr. Alberto Abad and Prof. Dr. Isabel Trancoso for having been always available, at any time and for having provided me brilliant solutions to the problems I encountered during the realization of this work. They always sustained me with solicitude, ready to offer their precious advices and their warmest smile.

Thanks also goes to all the friendly, gentle colleagues and roommates who I had the pleasure to meet during this experience. All of them have made their best, since the beginning, to make me feel comfortable. A special thanks goes to Angela, not only for her pleasant company, but also for having helped me to translate the abstract of this thesis in Portuguese.

For me this is also a great opportunity to express my gratefulness to the ones who shared with me these years of studies.

First, I wish to express all my gratitude to my beloved parents and uncles, for their understanding, endless care and for having always backed and encouraged me.

My deepest thanks is for my lifelong companion, Paolo. He has been near me in this long way providing me invaluable support both moral and professional. He helped me in overcoming so many difficulties. Thanks for having been always there, good and bad times.

Last, but not least, a special thanks goes to my friend Sara. She has been more than a sister to me, I always been able to count on her for any kind of help, just for a talk, or for a bier in our favorite pub.

This work was funded by the FCT project RIPD/ADA/109646/2009.
# Contents

1 Introduction

1.1 Context: The Vithea Project ........................................ 1
1.2 Objectives ......................................................... 3
1.3 Organization ....................................................... 3

2 Background on aphasia disorder and treatments .......................... 5

2.1 Classification of aphasia disorder .................................. 5
2.2 Aphasia treatment ................................................ 7

3 Speech and Language Technologies ..................................... 9

3.1 Automatic Speech Recognition ...................................... 9
3.2 Common problems in speech recognition ............................ 10
    3.2.1 Continuous Speech versus Isolated Word Recognition .... 10
    3.2.2 Speaker variability, Speaker dependent versus Speaker independent ...................................................... 11
    3.2.3 Environment variability ........................................ 11
3.3 Overview of a speech recognizer system and AUDIMUS recognizer ..................................................... 12
    3.3.1 Front-End ...................................................... 13
    3.3.2 Back-end ...................................................... 14
3.4 AUDIMUS within the context of the Vithea system ................. 18
    3.4.1 Corpus ....................................................... 18
    3.4.2 Keyword Spotting ........................................... 19

4 Open Source Software for distance learning ............................ 20

4.1 Introduction on distance learning evolution .......................... 20
4.2 Learning Management System ...................................... 21
    4.2.1 Benefits ...................................................... 24
    4.2.2 Drawbacks .................................................... 24
    4.2.3 Conclusions .................................................. 26
## List of Figures

3.1 Schematic structure of the basic steps of an automatic speech recognition system. ........................................... 12  
3.2 AUDIMUS system, details on Feature extraction. ............ 14  
3.3 AUDIMUS system, details on Acoustic Model. ................. 15  
3.4 AUDIMUS main blocks. ........................................... 18  

5.1 Struts 2 MVC ......................................................... 34  
5.2 Struts 2 Architecture ............................................. 36  
5.3 Class Diagram of Book and Publisher [Mak 10] ............... 39  
5.4 Entity-relationship (ER) diagram of Book and Publisher [Mak 10] ......................................................... 40  
5.5 The role of Hibernate in a Java application [Linwood 10] .... 43  
5.6 Overview of the Spring framework [Johnson 10] ............... 47  

6.1 Some images from the original Snodgrass and Vanderwart set [Aszekely 04]. .................................................. 52  
6.2 Architectural Overview of the Vithea System. .................. 53  
6.3 A capture image of the Client module interface that lists the preview of an exercise' questions ................................. 55  
6.4 A representation of the Vithea system ............................ 57  
6.5 Audio Recorder ..................................................... 63  
6.6 Web Application structure ........................................ 64  
6.7 Entity Relationship Diagram of Vithea - User Related table . 73  
6.8 Entity Relationship Diagram of Vithea - Exercises related table 74  
6.9 Entity Relationship Diagram of Vithea - Tracking related table 75
# Listings

5.1 Java code for selecting data from a table via Java DataBase Connectivity (JDBC) [Linwood 10] .................................. 41
5.2 Java code for selecting data from a table with Hibernate [Linwood 10] ................................................................. 42
5.3 Annotated Book class with Jpa standard ......................... 44
5.4 Excerpt of a Spring configuration file ......................... 48
6.1 Excerpt of a Tiles configuration file ......................... 59
6.2 Excerpt of Vithea’s main template ............................. 60
6.3 Excerpt of Vithea’s configuration file ......................... 60
6.4 Excerpt of Vithea’s web.xml configuration file ............. 65
6.5 Excerpt of Vithea’s applicationContext.xml configuration file ................................. 66
6.6 LoginUserBoImpl class ............................................. 66
6.7 Excerpt of Vithea’s web.xml configuration file ............. 68
6.8 Struts main configuration file .................................... 69
6.9 Spring configuration file for integration with Struts 2 .... 69
6.10 Excerpt of Vithea’s applicationContext.xml configuration file ................................. 70
6.11 Dao configuration .................................................. 71
6.12 Dao source code ................................................... 71
<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
<td>15</td>
</tr>
<tr>
<td>AOP</td>
<td>Aspect Oriented Programming</td>
<td>46</td>
</tr>
<tr>
<td>ASR</td>
<td>Automatic Speech Recognition</td>
<td>2</td>
</tr>
<tr>
<td>BN</td>
<td>Broadcast News</td>
<td>9</td>
</tr>
<tr>
<td>CSR</td>
<td>Continuous Speech Recognition</td>
<td>9</td>
</tr>
<tr>
<td>CVA</td>
<td>Cerebral Vascular Accident</td>
<td>2</td>
</tr>
<tr>
<td>DBMS</td>
<td>DataBase Management System</td>
<td>30</td>
</tr>
<tr>
<td>FA</td>
<td>False Alarm</td>
<td>77</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
<td>14</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
<td>31</td>
</tr>
<tr>
<td>IOC</td>
<td>Inversion of Control</td>
<td>32</td>
</tr>
<tr>
<td>IVR</td>
<td>Interactive Voice Response</td>
<td>9</td>
</tr>
<tr>
<td>IWR</td>
<td>Isolated Word Recognition</td>
<td>10</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
<td>31</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java DataBase Connectivity</td>
<td>viii</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistence Api</td>
<td>43</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
<td>32</td>
</tr>
<tr>
<td>LM</td>
<td>Language Model</td>
<td>17</td>
</tr>
<tr>
<td>LMS</td>
<td>Learning Management System</td>
<td>20</td>
</tr>
<tr>
<td>LVCSR</td>
<td>Large Vocabulary Continuous Speech Recognition</td>
<td>19</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron</td>
<td>15</td>
</tr>
<tr>
<td>MSG</td>
<td>Modulation Spectrogram</td>
<td>13</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
<td>31</td>
</tr>
<tr>
<td>ORM</td>
<td>Object-Relational Mapping</td>
<td>40</td>
</tr>
<tr>
<td>PLP</td>
<td>Perceptual Linear Predective</td>
<td>13</td>
</tr>
<tr>
<td>RASTA-PLP</td>
<td>log-RelAtive SpecTrAl PLP</td>
<td>13</td>
</tr>
<tr>
<td>SLT</td>
<td>Speech and Language Technologies</td>
<td>2</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
<td>37</td>
</tr>
<tr>
<td>TTS</td>
<td>Text to Speech</td>
<td>79</td>
</tr>
<tr>
<td><strong>WAF</strong></td>
<td>Web Application Framework</td>
<td>28</td>
</tr>
<tr>
<td><strong>WFST</strong></td>
<td>Weighted Finite State Transducer</td>
<td>17</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter aims at illustrating the motivations underlying the development of the Vithea system (see Section 1.1) and at specifying its main objectives (see Section 1.2). Finally, section 1.3 describes the structure of the remainder of this thesis.

1.1 Context: The Vithea Project

Communication is essential in our life. Every day we spend most of our time communicating, at work, at school, at home, at the restaurant. Most of us depend on their communication abilities to function in our society. Therefore, it appears clear how people who suffer from some kind of communication disorder belong to a greatly disadvantaged class. Disadvantages appear both in daily activities, in which they are incapacitated to express themselves in common tasks, as well as in the working area, in which they may find themselves completely excluded or restricted to lower level jobs. These limitations of work or lifestyle may result in a devastating psychological condition both for the affected individual, as well as for its family that struggles with the impossibility to communicate with his/her loved one. Moreover, communication disorders have a major impact on the economy. Both the cost of care as well as the degradation of the employment opportunities cause a huge expense to the national economy of a country.

Language disorders span from problems in producing speech sounds correctly or fluently, to difficulties in hearing comprehension or a combination of both. These problems may involve both syntax and semantics aspects of a language and they can affect both spoken and written language. Speech impediments may be caused by a neurological disorder, a brain injury, or could be a consequence of congenital physical impairments (i.e. cleft lip and
Aphasia is a speech disorder which comprises difficulties in both producing and comprehension of spoken or written language. It is caused by damage to one or more of the language areas of the brain, typically it occurs after brain injuries. Many times, the cause of the brain injury is a Cerebral Vascular Accident (CVA), but other causes could be brain tumors, brain infections and severe head injury due to car and work accidents. Unfortunately, nowadays the numerous cases of young individuals that have reported a brain injury due to car or work accidents, has lead to a gradual increasing of the number of individuals with aphasia caused by a CVA. Though, there are not statistical data about the number of individuals with language disorders, it is estimated that 200,000 new cases occur every year in the EU [Stachowiak 93].

Aphasia presents several different diagnosis and is therefore classified in different typologies, however an impairment in word retrieval is a common, if not universal, symptom of aphasia [Wilshire 00]. Regardless with a specific medical classifications, all aphasic individuals have problems in retrieving words for spoken production.

Besides, several studies have also demonstrated that constant and frequent therapy sessions have obtained positive effects with respect to poorer sessions in terms of hour-per-day or session-per-week [Bhogal 03].

In this context, an important consideration that needs to be kept into account is the progress made in the area of Speech and Language Technologies (SLT) which over the last years became truly significantly. Several works can be found in the literature, related both to help people with physical impairment in daily needs and also for diagnosis, assessment and treatment of speech diseases [Hawley 05, Maier 09, Castillo-Guerra 03, Yin 09]. Now this technology evolved at a point that its application in the rehabilitation of speech disorders will provide increased possibilities, thus contributing to both reduce the cost of the care of communication disorders as well as providing better rehabilitation, which in turns will increase the employment opportunities for people with such disorders.

The aforementioned key points are the compelling reasons which have lead to the development of the national Portuguese project VITHEA - Virtual Therapist for Aphasia Treatment.

The system will act as a "virtual therapist", simulating a conventional therapy session for the treatment of one of the most common aphasia disorder, namely word-recall. Specifically, the system will ask for the patient to recall the contents of a photo or a picture that is shown. With the integration of Automatic Speech Recognition (ASR) technology the system will recognize and validate the correctness of utterances produced by patients while interacting with the virtual therapist. An important benefit that the
program will bring is that therapy sessions now could be carried out in the comfort of own home, at any time which will cause an increase in the number of training hours and consequently we expect significant improvements in the rehabilitation.

1.2 Objectives

The goal of this work is to develop an accessible and effective software program that will act as a Virtual Therapist for treating speech and language disorder like aphasia. The program will be designed to immerse individuals in multimedia therapy sessions with the ultimate goal of achieving general improvements in language comprehension and production. In particular, we plan to develop:

- an administration module specifically designed for the clinicians. This module will allow to plan and develop therapy sessions in a simple and intuitive fashion. Besides, it will provide the possibility to monitor the frequency of usage of the system by the users.

- a client module specifically designed for the patients. This module will behave as a Virtual Therapist, guiding the user through the training exercises.

- an external component that will record user utterances and interact with an ASR system in order to evaluate the correctness of the patients’ answers to the proposed exercises.

The project addresses a great national need for inexpensive, intensive and extensive treatments for individuals with aphasia. It aims to be an accessible, affordable and easy way to use speech and language therapy by individuals with aphasia in the comfort of their homes. We hope that constant, intense training sessions will help to improve speech production and comprehension skills of individuals with aphasia.

1.3 Organization

The structure of this thesis aims to guide the reader from the motivations of this project, up to its fulfillment, explaining through the chapters the choices done in order to accomplish our objectives.

We start, in chapter 2, by briefly describing both the social and the economical conditions which involve language diseases like aphasia. Then,
we introduce different aphasia’s syndromes and one common disorder that they all share. Last, we examine various types of treatments and how the frequency of therapy sessions may influence the recovery.

In chapter 3 we focus on introducing the technology employed in an essential building block of the Vithea system, namely the ASR subsystem.

In chapter 4 we describe our first approach to develop the Virtual Therapist, in which several mainstream technologies of distance learning application were evaluated. Apparently the project we plan to develop seems to match in terms of requirements the features that Learning Management Systems offer. Eight different systems were studied and compared, analyzing their benefits and drawbacks. The results of this evaluation will reveal the inadequacy of these tools for our purposes.

Therefore, we decided to develop an entirely new application, starting from the ground up and following best software and architectural principles in order to lead to a modular, extensible program. Chapter 5 is devoted to describe the features of some of the most popular software frameworks, specifically designed to support the development of Web Applications. Particularly, three frameworks that will be used in the project, are described in depth.

Finally, chapter 6 describes the current implementation of the project, focusing both on the pedagogical approach used while developing the prototype as well on technical detail of the application itself.

Last, chapter 7 provides our conclusions and lists future enhancements that will be integrated within the project.
Chapter 2

Background on aphasia disorder and treatments

This chapter is devoted to provide a clear understanding of aphasia disorder. Different types of aphasia are briefly introduced in section 2.1, while different type of treatments are listed in section 2.2.

2.1 Classification of aphasia disorder

There are different types of aphasia. Each type can cause impairment that varies from mild to severe. With mild aphasia, the person may be able to converse yet have trouble finding the right word or understanding complex conversations. Severe aphasia limits the person’s ability to communicate. The person may say little and may not participate in or understand any conversation.

We can classify various aphasia syndromes by characterizing the speech output in two broad categories: fluent and non-fluent aphasia [Goodglass 93]. Fluent aphasia has normal articulation and rhythm of speech, but is deficient in meaning. Speech is characterized by a facility of articulation and many long runs of words combined using a variety of grammatical constructions. However, fluent speech is not equivalent to meaningful speech. Typically, there are word-finding problems that most affect nouns and picturable action words.

Non-fluent aphasic speech is slow and labored with short utterance length. The flow of speech is more or less impaired at the levels of speech initiation, the finding and sequencing of articulatory movements, and the production of grammatical sequences. Speech is choppy, interrupted, and awkwardly articulated.
Following the classification between fluent and non-fluent aphasia we list the major syndromes and their properties:

1) **Fluent**

   a) *Wernicke’s aphasia* is caused by damage to the temporal lobe of the brain, is one of the most common syndromes in fluent aphasia. People with Wernicke’s aphasia may speak in long sentences that have no meaning, add unnecessary words, and even create made-up words. For example, someone with Wernicke’s aphasia may say, "You know that smoodle pinkered and that I want to get him round and take care of him like you want before", meaning "The dog needs to go out so I will take him for a walk". They have poor auditory and reading comprehension, and fluent, but nonsensical, oral and written expression. Individuals with Wernicke’s aphasia usually have great difficulty understanding the speech of both themselves and others and are therefore often unaware of their mistakes.

   b) *Transcortical aphasia* presents similar deficits as in Wernicke’s aphasia, but repetition ability remains intact.

   c) *Conduction aphasia* is caused by deficits in the connections between the speech-comprehension and speech-production areas. This might be caused by damage to the arcuate fasciculus, the structure that transmits information between Wernicke’s area and Broca’s area. Auditory comprehension is near normal, and oral expression is fluent with occasional paraphasic errors. Repetition ability is poor.

   d) *Anomic aphasia* with anomic aphasia the individual may have difficulties naming certain words, linked by their grammatical type (e.g. difficulty naming verbs and not nouns) or by their semantic category (e.g. difficulty naming words relating to photography but nothing else) or a more general naming difficulty.

2) **Non-fluent**

   a) *Broca’s aphasia* is caused by damage to the frontal lobe of the brain. People with Broca’s aphasia may speak in short phrases that make sense but are produced with great effort. They often omit small words such as "is", "and", and "the". For example, a person with Broca’s aphasia may say, "Walk dog", meaning, "I will take the dog for a walk", or "book book two table", for "There are two books on the table". People with Broca’s aphasia typically understand the speech of others fairly well. Because of this, they are often aware of their difficulties and can become easily frustrated.
b) Global aphasia presents severe communication difficulties, individuals with global aphasia will be extremely limited in their ability to speak or comprehend language. They may be totally non-verbal, and/or only use facial expressions and gestures to communicate. It is associated with right hemiparesis, meaning that there can be paralysis of the patient’s right face and arm.

c) Transcortical Motor aphasia presents similar deficits as Broca’s aphasia, except repetition ability remains intact. Auditory comprehension is generally fine for simple conversations, but declines rapidly for more complex conversations. It is associated with right hemiparesis, meaning that there can be paralysis of the patient’s right face and arm.

However, regardless from the specific syndrome, a common symptom in all types of aphasia, even if not universal, is an impairment in word retrieval. Regardless of diagnostic classification, virtually all aphasic patients have difficulty retrieving words for spoken production, and this is most clearly observed in their generally poor performance on tasks such as picture naming [Wilshire 00].

2.2 Aphasia treatment

In some cases, a person will completely recover from aphasia without treatment. This type of spontaneous recovery usually occurs following a type of stroke in which blood flow to the brain is temporarily interrupted but quickly restored, called a transient ischemic attack. In these circumstances, language abilities may return in a few hours or a few days. For most cases, however, language recovery is not as quick or as complete. While many people with aphasia experience partial spontaneous recovery, in which some language abilities return a few days to a month after the brain injury, some residual disorders typically remain. In these instances, most clinicians would recommend speech-language therapy. The recovery process usually continues over a two-year period, although clinicians believe that the most effective treatment begins early in the recovery process.

There are multiple modalities of speech therapy. The most commonly used techniques are output focused such as the stimulation method and the Melodical Intonation Therapy (MIT). MIT is a formal, hierarchically structured treatment program based on the assumption that the stress, intonation, and melodic patterns of language output are controlled primarily by the right hemisphere and, thus, are available for use in the individual with aphasia with left hemisphere damage [Albert 94].
Other methods are linguistic-oriented learning approaches such as the lexical-semantic therapy or the mapping technique for the treatment of agrammatism. Still, other techniques such as Promoting Aphasics’ Communicative Effectiveness (PACE), focus on enhancing communicative ability, non-verbal as well as verbal, in pragmatically realistic settings [Davis 85]. Several non-verbal methods for the treatment of severe global aphasics rely on computer-aided therapy such as the visual analogue communication, iconic communication, visual action and drawing therapies are currently used [Sarno 81, Albert 98]. An example is Computerized visual communication (or C-VIC) designed as an alternative communication system for patients with severe aphasia and is based on the notion that those with severe aphasia can learn an alternative symbol system and can use this alternative system to communicate [Martinn 98].

Furthermore, although there exists such an extended list of treatments specifically thought to recover a particular disorder caused by aphasia, one class of treatment especially important is the one devoted to help improving word retrieval problems, since as we saw it is one of the most common residual disorder in all aphasia syndromes. Naming abilities problems are typically treated with semantic exercises like Naming Objects or Naming common actions where commonly the patient is asked to name a subject represented in a picture [Adlam 06].

However, although the type of aphasia and its treatment are major determinants of the outcome and of the pattern of evolution, many studies have investigated and proved how intensity of aphasia therapy may positive influence aphasia recovery. Intensity of speech and language therapy measured in terms of length of therapy and hours of therapy may led to satisfactory results recovery with respect to lesser or shorter therapy sessions [Ferro 99, Bhogal 03].
Chapter 3

Speech and Language Technologies

This chapter explains the technologies that compose an Automatic Speech Recognition system and describes the role it plays within the Vithea system. In section 3.1 we describe how recent progress in the area of Speech and Language Technologies (SLT) may provide significant support in the recovery of people with language disorder issues. Common problems faced in this particular field are explained in section 3.2. In section 3.3 we provide a brief overview of the main building blocks which make up an ASR system, particularly we deepen in the description of AUDIMUS, the ASR system developed by the L2F group that has been integrated into the Vithea system. Last section 3.4 is devoted to explain how automatic word retrieval task, the main common residual impairment in aphasia, is carried out.

3.1 Automatic Speech Recognition

In the area of SLT one field that has gathered significant improvements in the last years is ASR. ASR can be described as the process of converting into text the speech uttered by human speakers. Speech recognition may range from understanding simple words or digits, like in Interactive Voice Response (IVR) where communication occurs between humans and computers through short voice commands, to Continuous Speech Recognition (CSR), like in dictation systems. ASR applications’ list is rather extensive, besides IVR and dictation systems it can be applied to telematics (e.g. vehicle Navigation Systems), multimodal interaction, hands-free computing, robotics, transcription of Broadcast News (BN) [Meinedo 08], pronunciation evaluation in computer-aided language learning applications [Marujo 09], automatic
translation, domotics, and further more. However, one of the most remark-
able application areas for ASR is helping people with disabilities. During the
last decade, the development of more sophisticated techniques for analyzing
incoming speech combined with the increased processing power of computers
has resulted in improved recognition performance. Consequently, speech rec-
ognizers have become more affordable and more accurate. For many people
with wide-ranging disabilities, speech recognition has become an extremely
viable option. People with different types of speech impediments, physical
disabilities and even ones with memory and concentration problems may
benefit from this technology.

3.2 Common problems in speech recognition

Humans use more than their ears when listening, they use the knowledge
they have about the speaker and the subject of the speech. Words are not
arbitrarily sequenced together, there is a grammatical structure that humans
use to predict words not yet spoken. Furthermore, idioms and how we "usu-
ally" say things makes prediction even easier. In ASR there only exists the
speech signal. As we will see in section 3.3 ASR of course uses a model for the
grammatical structure and some statistical model to improve prediction, but
the problem of how to model world knowledge, the knowledge of the speaker
and comprehensive knowledge of the surrounding context, opens important
considerations.

The level of complexity of an ASR may vary according to several fac-
tors, some of them depend on the purpose of the recognizer system. In the
following we list briefly the main problems that constitute the recognition
task.

3.2.1 Continuous Speech versus Isolated Word Recogn-
nition

The speech signal is different if input is given with isolated words or the
speech is continuous. An Isolated Word Recognition (IWR) system operates
on single words at a time - requiring a pause between saying each word.
This is the simplest form of recognition to perform because the end points
are easier to find and the pronunciation of a word tends not to affect others.
Thus, because the occurrences of words are more consistent they are easier to
recognize. A CSR system operates on speech in which words are connected
together, i.e. not separated by pauses. Continuous speech is more difficult to
handle because of a variety of effects. First, it is difficult to find the start and
end points of words. Another problem is "coarticulation". The production of each phoneme is affected by the production of surrounding phonemes, and similarly the start and end of words are affected by the preceding and following words.

3.2.2 Speaker variability, Speaker dependent versus Speaker independent

Every individual speaker is different due to its unique physical body and personality. The speech he or she produces is influenced both from physical characteristics (i.e. the anatomy of vocal tract, sex) and other factors like regional and social dialects and education. These variations are known as interspeaker differences and lead to the obvious consequence that the speech patterns of one person can be totally different from those of another person. Besides interspeaker, there exist also intraspeaker differences. Indeed, the same speaker could be unable to exactly reproduce the same utterance due to a particular emotional state.

A speaker independent system is developed to operate for any speaker of a particular language (e.g. American English). These systems are the most difficult to develop: they typically require more than 500 speakers to build a combined model, also accuracy is lower compared to speaker dependent systems.

A speaker dependent system is developed to operate for a single speaker. These systems are usually easier to develop and more accurate, but not as flexible as speaker independent systems. Their main disadvantage is that it takes time to collect speaker-dependent data, which may be impractical for some applications.

3.2.3 Environment variability

The environment also causes variability and must be taken into account. Speech is uttered in an environment surrounded by other sounds, other human speakers in the background, a clock ticking, a computer humming and many others. This is usually called noise, unwanted information in the speech signal. In ASR these noises have to be identified and filtered out from the speech signal. Another kind of noise is the echo effect, which is the speech signal bounced on some surrounding object, and that arrives in the microphone a few milliseconds later. If the place in which the speech signal has been produced is strongly echoing, then this may give raise to a phenomenon called reverberation, which may last even as long as seconds.
3.3 Overview of a speech recognizer system and AUDIMUS recognizer

For several years L²F has dedicated its research efforts to the development of ASR systems, achieving so a great experience in the area. Through the participation in several projects, national and international, the group dedicated first on ASR systems for the English language with a major focus in speaker-adaptation techniques [Neto 98, Neto 96]. Then, the collected knowledge has been used to develop ASR systems for the European Portuguese language. AUDIMUS is the outcome of this work, it has been successfully adopted for different tasks like dictation, telephone, Broadcast News, portable devices and many others. In the following we provide an overview of the main building blocks of a speech recognizer system, describing for each of them how AUDIMUS achieves this task. As shown in figure 3.1 when the speech signal is received it is first converted into a sequence of feature vectors, which are used to feed the speech decoder. The decoder seeks for the best match between a sequence of features and every possible sequence of acoustic classes, using the available information from the acoustic and language models, which are typically obtained in a training phase prior to the recognition step.

![Figure 3.1: Schematic structure of the basic steps of an automatic speech recognition system.](image)

In automatic speech recognition feature vectors provided by the front-end are matched with reference pattern called acoustic models. Given the sequence of feature vectors defined as $X = x_1 x_2 \ldots x_T$ where $x_t$ is the observed feature vector at time instant $t$, the goal of speech recognition is to find out the corresponding word sequence $W = w_1 w_2 \ldots w_n$ that has the maximum posterior probability $P(W|X)$. That is, the word sequence that
has more likely been said given a concrete sequence of observation vectors:

\[ W = \arg \max_w P(W|X) = \arg \max_w \frac{P(W)P(X|W)}{P(X)} \]  

(3.1)

Since the maximization is carried out with the observation \( X \) fixed, the denominator can be ignored, resulting:

\[ W = \arg \max_w P(W)P(X|W) \]  

(3.2)

### 3.3.1 Front-End

Speech recognition systems do not actually perform the recognition or decoding step directly on the speech signal. Rather, the speech waveform is divided into short frames of samples, which are converted to a meaningful set of features. The duration of the frames is selected so that the speech waveform can be regarded as being stationary. In addition to this transformation, some pre-processing techniques are applied to the waveform signal in order to enhance it and to better prepare it for the speech recognition. The whole set of these steps applied to the speech signal in order to obtain the final feature vector is commonly known as the front-end of the speech recognizer.

#### Feature extraction

In the Feature Extraction step, the sampled speech signal is parametrized. The goal is to extract a number of parameters (‘features’) from each frame of the signal containing the relevant speech information and being robust to acoustic variations and sensitive to linguistic context. More in detail, features should be robust against noise and factors that are irrelevant for the recognition process, also features that are discriminant and allow to distinguish between different linguistic units (e.g., phones) are required.

As shown in figure 3.2, AUDIMUS system uses three different feature sets:

- Perceptual Linear Predective (PLP) [Hermansky 90]
- log-RelAtive SpecTrAl PLP (RASTA-PLP) [Hermansky 92]
- Modulation SpectroGram (MSG) [Kingsbury 98]

This merged approach has proved being more efficient and robust with respect to using one of the feature individually [Meneido 00]. This is explained by the integration of the advantages of these three feature sets: the inclusion of the attributes of the psychological processes of human hearing into the
analysis used with PLP [Hermansky 90, Jamaati 08] makes the speech perception more human-like, the compensation for linear channel distortions provided by RASTA-PLP [Hermansky 92], the improved performance in terms of stability provided by MSG [Kingsbury 98] in the presence of acoustic interferences, like high levels of background noise and reverberation. [Anatol Koller 10]

### 3.3.2 Back-end

The goal of a back-end stage of a recognizer is to do a mapping of the speech vectors provided by the front-end and the wanted underlying sequence of acoustic classes modeling concrete symbols (phonemes, letters, words...). Both acoustic pattern matching and knowledge about language are important in recognizing and understanding natural speech. Lexical knowledge (i.e., vocabulary definition and word pronunciation) is required, as are the syntax and semantics of the language. In addition, knowledge of the pragmatics of language can be important to achieve the goal of spoken language understanding systems.

**Acoustic Modeling**

Acoustic Modeling is arguably the central part of any speech recognition system, it plays a critical role in improving accuracy. The practical challenge is how to build accurate acoustic models, $P(X|W)$ that can truly reflect the spoken language to be recognized. $P(X|W)$ should take into account the problems we discussed in earlier section 3.2, namely speaker variations, environment variations, pronunciation variations.

An extended and successful statistical parametric approach to speech recognition is the Hidden Markov Model (HMM) paradigm [Rabiner 89, Ra-
biner 93] that supports both acoustic and temporal modeling. The underlying assumption of the HMM is that the data samples can be well characterized as a parametric random process, and the parameters of the stochastic process can be estimated in a precise and well-defined framework [Huang 01]. HMMs model the sequence of feature vectors as a piecewise stationary process. An utterance $X = x_1, \ldots, x_n, \ldots, x_N$ is modeled as a succession of discrete stationary states $Q = q_1, \ldots, q_k, \ldots, q_K$, $K < N$, with instantaneous transitions between these states. An HMM is typically defined as a stochastic finite state automaton, usually with a left-to-right topology. It is called "hidden" Markov model because the underlying stochastic process (the sequence of states) is not directly observable, but still affects the observed sequence of acoustic features.

Alternatively, Artificial Neural Networks (ANN) have been proposed as an efficient approach to acoustic modeling [Tebelskis 95]. Although for the past thirty years ANNs have been used for difficult problems in pattern recognition, more recently many researchers have shown that these nets can be used to estimate probabilities that are useful for speech recognition. Particularly, here we will only briefly introduce Multilayer Perceptron (MLP)s, which are the most common ANN used for speech recognition. Typically, MLPs have a layered feedforward architecture with an input layer, zero or more hidden layers, and one output layer [Bourlard 94,Bourlard 93].

ANN-HMM hybrid systems have been focus of research in order to combine the strengths of the two approaches. Systems based on this hybrid approach have performed very well on large vocabulary continuous speech recognition.

![AUDIMUS System Diagram](image)

**Figure 3.3**: AUDIMUS system, details on Acoustic Model.

AUDIMUS is a hybrid recognizer that follows the aforementioned connectionist approach. It combines the temporal modeling capacity of Hidden
Markov Models (HMMs) with the pattern discriminative classification of multilayer perceptrons (MLPs). The Markov process here is used to model the basic temporal nature of the speech signal, while artificial neural network is used to estimate posterior phone probabilities given the acoustic data at each frame. Each MLP outputs a phone probability, these are all combined using an average in the log-probability domain. Figure 3.3 illustrates the details of AUDIMUS acoustic models.

The current version of AUDIMUS, integrated within the context of the project at the time of writing this thesis, uses an acoustic model trained with 57 hours of BN and 58 hours of mixed fixed-telephone and mobile-telephone data [Abad 08].

**Language Model and Lexical knowledge**

Knowledge of the rules of a language, the way in which words are connected together into phrases and what people are likely to say in particular contexts are important to achieve the goal of spoken language understanding systems. This is the job of the language model. Informally, its aim is to predict the likelihood of specific words occurring one after another in a certain language. In a more formal description, the probability of the i-th word following the (i – 1) previous words is defined as $P(w_i|w_1, w_2, \ldots, w_{i-1})$. Therefore, the probability distribution $P(W)$ over word strings $W = w_1, w_2, \ldots, w_n$, that reflects how frequently a string $W$ occurs as a sentence can be decomposed as follows:

$$P(W) = P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, \ldots, w_{n-1}) = \prod_{i=1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1}) \quad (3.3)$$

In Eq. 3.3, the choice of $w_i$ thus depends on the entire past history of the input. For a vocabulary of size $v$ there are $v^{i-1}$ different histories given a string of $i – 1$ words and so, to specify $P(w_i|w_1, w_2, \ldots, w_{i-1})$ completely, $v^i$ values would have to be estimated. In reality the probabilities $P(w_i|w_1, w_2, \ldots, w_{i-1})$ are impossible to estimate for even moderate values of $i$, since most histories $w_1, w_2, \ldots, w_{i-1}$ are unique or have occurred only a few times. A practical solution to the above problems is to assume that $P(w_i|w_1, w_2, \ldots, w_{i-1})$ depends only on some equivalence classes. The equivalence class can be simply based on the $N$ previous words $w_{i-N+1}, w_{i-N+2}, \ldots, w_{i-1}$. This leads to an n-gram language
model. If the word depends on the previous two words, we have a trigram: $P(w_i|w_{i-2},w_{i-1})$. Similarly, we can have unigram: $P(w_i)s$, or bigram: $P(w_i|w_{i-1})$ language models [Huang 01].

The current version of AUDIMUS used within the context of the project, uses a Language Model (LM) which is a statistically 4-gram model with 100k 1-gram, 7.5M 2-gram, 14M 3-gram and 7.9M 4-gram that result from the interpolation of several LMs [Meinedo 10].

In addition to language knowledge, Lexical knowledge (i.e., vocabulary definition and word pronunciation) is also required. Basically, a dictionary of pronunciations defines which combination of phones give valid words for the recognition. It can contain information about different pronunciation variants of the same word.

**Decoder**

The decoding process of a speech recognizer is to find a sequence of words whose corresponding acoustic and language models best match the input signal. Therefore, the process of such a decoding process with trained acoustic and language models is often referred to as just a search process. Its complexity depends both if IWR versus CSR is used, as well as from the unit of the acoustic model. With IWR word boundaries are known, the word with highest forward probability is chosen as the recognized word and the problem search problem becomes a simple pattern recognition problem. While search in CSR is more complicated since the search algorithm has to consider the possibility of each word starting at any arbitrary time frame. Moreover, the unit of acoustic model $P(X|W)$ is not always a word model. For large vocabulary speech recognition systems, sub-phonetic models, which include phonemes, diphones, triphones, or other sub-word models are often used. When sub-word models are used, the word model $P(X|W)$ is then obtained by concatenating the sub-word models according to the pronunciation transcription of the words in a lexicon or dictionary. When word models are available, speech recognition becomes a search problem. The goal for speech recognition is thus to find a sequence of word models that best describes the input waveform against the word models. As neither the number of words nor the boundary of each word or phoneme in the input waveform is known, appropriate search strategies to deal with these variable-length nonstationary patterns are extremely important [Huang 01].

The AUDIMUS decoder is based on the Weighted Finite State
Transducer (WFST) approach to large vocabulary speech recognition [Mohri 02]. In this approach, the posterior phone probabilities from the acoustic model used in the HMMs are mapped to words. For this purpose the acoustic model topology $H$, the pronunciation lexicon $L$ and the language model $G$ are represented as transducers. The composition of the various transducers forms the final WFST, what leads to a search space of $H \circ L \circ G$ [Caseiro 03]. Figure 3.4 shows AUDIMUS system’s components all together.

![AUDIMUS System Components](image)

Figure 3.4: AUDIMUS main blocks.

### 3.4 AUDIMUS within the context of the Vithea system

This section overviews the AUDIMUS system, focusing the description of its main building blocks involved to accomplish the task of word naming recognition. These information have also been described in the paper ”Automatic word naming recognition for aphasia treatment”, further details can be found in [Abad 11].

#### 3.4.1 Corpus

A Speech corpus is a database comprising speech audio file and text transcription. It is used to train the Acoustic Models. Large vocabulary continuous speech recognizer requires large amounts of transcribed data to achieve robustly trained acoustic models. Nevertheless, manual transcription is an
expensive, time consuming and tedious job. Also, public sources of manual transcribed data are a rare in the case of less studied variety [Anatol Koller 10]. Thus, initially AUDIMUS has been bootstrapped with a sub-set of the already available speech databases for generating a test corpus similar to the operation conditions expected in the final prototype. Two sub-sets of the Portuguese SpeechDat II [SpeechDat 05] corpus consisting of word spotting phrases using embedded keywords have been selected for assessment of word verification systems: a development set with 3334 utterances and an evaluation set with 481 utterances. Then, a reduced speech corpus composed by data collected during therapy session of eight different aphasic patients has been used to assess the robustness of the proposed methods.

3.4.2 Keyword Spotting

As mentioned in chapter 2 section 2.1 the difficulty to recall words or names is the most common feature of aphasia and in some cases it can be the only residual defect after rehabilitation [Wilshire 00]. As we will see in chapter 6 this kind of disorder is treated with word naming-like exercises. Therefore, we refer to word verification as the task that performs the evaluation of the utterances spoken by the patients, in a similar way to the role of the therapist in a rehabilitation’s session. This task consists in checking whether a word $W$ exists in a given speech segment $S$ or not. Although it may seem an easy issue to solve, similar to IWR, actually it should be also take into account that interactions with aphasic patients present a considerable amount of hesitations, doubts and repetitions. These disturbing factors do not permit to apply easier approaches, instead they conduct to a method known as Keyword Spotting. This technique aims at detecting a small set of keywords from a continuous speech. Keyword spotting approaches are mainly classified into two categories: one based on the acoustic match of the audio data with keyword models in contrast to a background model, and the other one based on Large Vocabulary Continuous Speech Recognition (LVCSR) [Szöke 05]. Acoustic approaches basically extend the IWR framework by incorporating an alternative competing model to the list of keywords generally known as background. The latter is needed to separate keywords from non-keywords as the aim of these systems is to obtain the highest possible keyword detection rate and minimize the number of false alarms or false rejection. The methods based on LVCSR search for the target keywords in the recognition result, usually in lattices or confusion networks, since they allow improved performances compared to searching in the raw output result [Abad 11].
Chapter 4

Open Source Software for distance learning

In this chapter we analyze new technologies that have spread in the last years in distance learning world, focusing our interest on Open Source Learning Management System (LMS). Indeed, the requirements of the Vithea system seem to well match with the features offered by these software applications. Eight different LMSs will be analyzed to study in detail the functionality they provide (section 4.2). The analysis will reveal benefits (section 4.2.1) and drawbacks (section 4.2.2) of these technologies, highlighting how powerful they can be, although also present important disadvantages for our purpose. Our final conclusions are presented in section 4.2.3

4.1 Introduction on distance learning evolution

In the last decade, the effect of Internet usage in education has increased gradually and new technologies have improved distance learning. Based on these new technologies, learning environments are now able to provide a wide range of educational alternatives. Distance learning is one of these alternatives which became attractive both for allowing students and instructors being physically in different locations at different time, as well as for the improved flexibility with respect to place and time constraints.

However, as it is widely known, distance learning is costly, and cost-effectiveness becomes more important as the institutions become large-scale providers of distance education. Thus, in a distance learning process, open source software can be used in many different phases such as application software that performs learning content preparation and in LMS which provides
learning content presentation in a web based environment [Aydin 10].

Although these tools appear a ready-to-use solution, it is really important to examine the whole picture when we select, set up and make an LMS ready for the learning process. Even if such LMS has all pedagogical tools we need, it could happen that it cannot be successfully exploited because no one is able to use it. Selecting an LMS therefore means selecting the right framework, method (or several methods) and multiple criteria of evaluation that seem important. In the following some frameworks and their methodology are presented.

4.2 Learning Management System

An LMS is a "framework that tracks, supports, manages and measures e-Learning activities". LMSs typically provide a wide set of functionality to support students’ learning such as file storage, forums, calendar, news, mail, submission management system, groups surveys, organization, assessments, FAQs or scheduling.

Keeping in mind the purpose behind the Vithea system, we have still not stressed how much the project is related to the distance learning application area. We have already highlighted the important advantage for the patient of doing the training sessions in an environment comfortable to him/her own habits at the time more suitable for him. However, what we did not stress are the requirements of tracking every user session, (i.e storing user utterances, user results), as well as system information utilization. A quick approach to LMS technology apparently reveals how well they fit all these needs, providing a ready-to-use solution which already implements the above listed requirements and much more functionality. Indeed, these systems typically include access control, provision of learning content, communication tools and administration of user groups, aiding teachers in establishing a learning environment that operates on similar objectives to traditional methods of education.

Thus, we started evaluating currently widespread solutions. Among the many software available, we identified a list of candidates for further evaluation: Atutor, Chamilo, Claroline, eFront, Ilias, Moodle, Olat, Sakai.

Atutor (www.atutor.ca)
Atutor is designed with accessibility and adaptability in mind. It complies with the W3C WCAG 1.0 accessibility specifications at the AA+ level. The software is cited as unique for its accessibility features, - useful to visually-impaired and disabled learners - and for its suitability for educational use
according to software evaluation criteria established by The American Society for Training and Development.
License: GPL - GNU General Public License
Designed by: Adaptive Technology Resource Center at the University of Toronto.
Developed with: PHP

Chamilo (www.chamilo.org)
The Chamilo project aims at ensuring the availability and quality of education at a reduced cost, through the distribution of its software free of charge, the improvement of its interface for devices portability and the provision of a free access public e-learning campus.
License: GPL - GNU General Public License
Designed by: Chamilo Association - a legally registered non-profit association under Belgian laws
Developed with: PHP

Claroline (www.claroline.net)
Claroline is a collaborative eLearning and eWorking platform allowing hundreds of organizations worldwide, ranging from universities to schools and from companies to associations, to create and administer courses and collaboration spaces over the web. Claroline has been developed following teacher’s pedagogical experiences and needs. The Claroline platform is organized around the concept of spaces related to a course or a pedagogical activity. Each space provides a list of tools that enable creation of learning contents, management of training activities, and interaction with other students.
License: GPL - GNU General Public License
Designed by: University Institute of Pedagogy and Multimedia from the UCL, and the CERDECAM (Research Center of the ECAM, Belgium).
Developed with: PHP

eFront (www.efrontlearning.net)
eFront is designed to assist with the creation of online learning communities while offering various opportunities for collaboration and interaction through an icon-based user interface. eFront claims to be an easy to use e-learning and "human capital development" system, making it suitable for both company and educational usage.
License: CPAL - Common Public Attribution License.
Designed by: Epignosis Ltd - an e-learning company based in Greece.
Developed with: PHP
Ilias (www.ilias.de)
Ilias allows users to create, edit and publish learning and teaching material. It offers a cooperative learning environment where every user has a comprehensive personal desktop to use the system, keep notes, bookmarks, create working groups and many others functionality. Ilias has a flexible test system where time based tests can be created with multiple choice, single choice, allocation questions, cloze questions (free text, select box), ordering, matching, hot spot and more question types.
License: GPL - GNU General Public License.
Designed by: Faculty of business administration, economics and social sciences at the University of Cologne in Germany.
Developed with: PHP

Moodle (moodle.org)
Moodle stands for Modular Object-Oriented Dynamic Learning Environment. Moodle describes itself as a course management system to help educators create "effective online learning communities". Moodle is designed to support pedagogies based on social constructionist approach to education, emphasizing that learners (and not just teachers) can contribute to the educational experience.
License: GPLv2 - General Public License (GPL) Version 2.
Designed by: Martin Dougiamas - a graduate degrees in computer science and education.
Developed with: PHP

Olat (www.olat.org)
Olat stands for Online Learning And Training. Olat supports any kind of online learning, teaching, and tutoring with few didactic restrictions. It has been developed since 1999, most recent version provides full scalability - meaning Olat can be run on a cluster of server - and a new and improved layout based on a usability evaluation.
License: Apache License 2.0
Designed by: University of Zurich.
Developed with: Java

Sakai (sakaiproject.org)
Sakai is a powerful yet flexible solution that supports not only teaching and learning but also research and administrative collaboration.
Sakai’s core tools like forums, chat rooms, and message center can be augmented with tools designed for a particular application like assignments,
syllabus, and WebDAV. Using the application, it is possible to create courses, manage assignments, share documents, prepare exams. And, as a result, grade information can be calculate, stored and distributed.

License: Educational Community License
Designed by: A community source software founded by the University of Michigan, Indiana University, MIT, Stanford, the uPortal Consortium, and the Open Knowledge Initiative (OKI) with the support of the Andrew W. Mellon Foundation.
Developed with: Java

In order to evaluate the effectiveness and feasibility of this kind of software for distance learning and lastly, find a possible candidate for the project, we compared the main features of the aforementioned LMSs. Table 4.1 provides a summary of the features analyzed.

4.2.1 Benefits

As we have already learned, probably the best contribution given by LMSs is the possibility to bring knowledge everywhere, at any time, affordable for anyone who may access to a PC. The purpose of these educational tools is to promote learning, quite often using a pedagogical approach to allow a better organization of the core material and to facilitate meaningful learning. Furthermore, LMS enables teachers to manage large numbers of students: an LMS can be used to assist just one teacher to manage one course, or can be used to manage the e-learning content and delivery for several teachers and thousands of students.

Thus, the major advantages of LMSs for e-Learning include bringing an improvement in the speed of change and quality of performance, providing access to resources and guidance to deal with educational change, tracking detailed information about individual learners, making the system highly beneficial to large online environments [Hall 01].

4.2.2 Drawbacks

LMSs obviously have disadvantages too. Unfortunately we observe that, no matter how useful an LMS can be, there are some limitations: they are not a universal remedy for managing everything related to e-learning. All LMSs are not appropriate for all environments and all users. An LMS offers so many features and options that it can be very confusing for users to use.

The main problem noticed with LMSs, is indeed a simple concept: they try to do everything, but one tool can not do all without losing functionality.
Table 4.1: LMS features

<table>
<thead>
<tr>
<th>General System Features</th>
<th>Atutor</th>
<th>Chamilo</th>
<th>Claroline</th>
<th>eFront</th>
<th>Ilias</th>
<th>moodle</th>
<th>Olat</th>
<th>Sakai</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-language user interface</td>
<td>A*</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Encountered Bug Various Nature</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td>Automatic email notification</td>
<td>A*</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>User Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User roles</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Custom user roles creation</td>
<td>N</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Self-registration and self-enrollment</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Registration and enrollment by</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>administrator approval</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual registration and enrollment</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>of users by administrators</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal folder</td>
<td>A*</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Course Materials</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Upload and manage documents in</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>different formats</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test / Survey</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create surveys and tests</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Different question types</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Add media files into the questions</td>
<td>A*</td>
<td>A*</td>
<td>A*</td>
<td>A**</td>
<td>A**</td>
<td>A*</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Export test result</td>
<td>N</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>A*</td>
</tr>
<tr>
<td>User Performance Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Track user performance</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Track user progress</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Track users activity statistics</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>Track the learning materials usage</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A*</td>
</tr>
<tr>
<td>statistics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reporting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View reports</td>
<td>N</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>A*</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Configure criteria for reports</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Export reports</td>
<td>N</td>
<td>A*</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

Legend:
A Available
A* Available with restrictions
A** Available with high restrictions
N Not Available
The more feature-rich an individual tool becomes, the more it loses its usefulness to the average user. Simpler tools, with the intent of performing one task seem to be easier for end users to understand. Thus, should not be very surprising if, generally, the initial reactions to a first approach with the interface of these tools is confusion. The most disorienting challenge is figuring out how the interface works and where to get the information needed.

Last, using a structured tool like an LMS may dictate the nature of interaction (instructor-learner, learner-learner, learner-content), essentially it determines what an instructor can do [Siemens 04, Unesco 06].

### 4.2.3 Conclusions

The aforementioned disadvantages unfortunately were also verified and confirmed by our study. Besides these, another important parameter that we took into account was the possibility for an LMS to be customized; that is, the adaptability of the framework to our needs.

The Vithea system needs an easy to use, appealing interface. Quite often patients victim of a CVA experience motor deficiencies such as reduced arm movements. This means the appearance of the system should be really simple, avoiding as much as possible a large sequence of operations to be performed before reaching the desired functionality. Most LMSs do not allow this simplification, as they provide a rich set of features (i.e agenda, mail, contacts) as soon as a user is logged-in the application and many of these (unnecessary for our purposes) functionality can not be easily removed.

Moreover, our needs for customization extend to these functionalities that these tools provide in terms of training exercises. We experienced several differences between the evaluated LMSs, but we can state that they all present various limitations in terms of multimedia files that are allowed to be used with the questions of the tests. The problem of integrating an external module which has to play the important task of recording users utterance and then communicating with an ASR system was also carefully considered. Unfortunately, almost none of them would have enabled an easy solution for achieving this goal. Our experience highlighted that it would have been extremely complex to customize the evaluated frameworks to meet the Vithea project requirements without introducing major structural changes to the code.

Our users belong to an average population that is not necessarily accustomed with the use of computer and even less with these tools, which in most cases are intended to spread out knowledge in environments such as universities or huge organizations. This means that our users may lack the technical skills necessary to work with an LMS and they are not interested in
making the extra effort of learning how to use the system in order to being then capable of performing the recovery exercises. All this, would inevitably result in a loss of motivation from their part.

Therefore, considered the conclusions that come from this study we have definitely opted for building from the ground up a modular application which will totally adheres to our requirements.
Chapter 5

Open Source Web application frameworks

This Chapter is devoted to overview the main Open Source Web Application Framework (WAF)s that have been integrated in the Vithea system.

We start, in Section 5.1, by highlighting the motivations underlying the recent proliferation of Web Application Frameworks, namely the key advantages that they provide in the design and implementation of Web Applications, as well as the typical architectural organization -from both a hardware and software perspective- of this class of applications.

Then, in Section 5.2, we describe the three main WAFs that were exploited to build the Vithea system, namely Struts 2 (see Section 5.2.1), Hibernate (see Section 5.2.2) and Spring (see Section 5.2.3).

5.1 Overview of Web Application Frameworks

In information systems environment, a framework is a defined support structure in which other software applications can be organized and developed. A framework may include support programs, code libraries, a scripting language, common services, interfaces, or other software packages utilities to help developing and glue together the different components of a software application. A software framework is a reusable design and building block for a software system.

A software framework consists of frozen spots and hot spots. The frozen spots define the overall architecture of a software system - its basic components and the relationships between them. These remain unchanged (frozen) in any instantiation of the application framework. On the other hand, hot
spots represent those parts of the software framework that are specific to individual software systems. Hot spots are designed to be generic. In other words, they can be adapted to the needs of the application under development. Software frameworks define the places in the architecture where adaptations for specific functionality should be made - the hot spots. In an object-oriented environment, a framework consists of abstract and concrete classes. Instantiation of such a framework consists of composing and sub-classing the existing classes [Shan 06].

Application frameworks offer a variety of advantages:

- A framework tries to make generalizations about the common tasks and workflow of a specific domain.

- Using code which has already been built, tested, and used by other programmers increases reliability and reduces the amount of time, effort, and resources required to develop and maintain applications.

- Frameworks can provide security features which are often required for a common class of applications. This provides every application written with the framework to benefit from the added security without the extra time and cost of developing it.

- By handling "lower level" tasks, frameworks can assist with code modularity. Business logic, for example, can remain in the application while the ordinary tasks of database connectivity and handling user logins can be handled separately in the framework.

- Frameworks can assist in programming to design patterns and general best practices.

The evolution that happened in World Wide Web brought a totally new class of web application ranging from enriched dynamic web sites to more sophisticated collaboration tools and integrated services which now are available online. Web Application nowadays have became popular to such an extent that, over the last years there has been a great involvement, specially from the Open Source community, in the designing of frameworks specialized for such applications. This gave origin to Web Application Framework, a reusable, skeletal, semi-complete modular platform that can be specialized to produce custom web applications. A web application framework is, indeed, specifically designed to help building web applications.

Normally frameworks for Web Applications take into account the typical architecture of this class of applications, generally structured according to
the multi-tier architectural pattern. In the following we describe the core building blocks which make up a web application:

- Web Browser: the top layer of most web applications, which directly interacts with the user. This is the one component over which web application developers have the least control, as on a public web server any version of any web browser may try to connect and interact. Therefore standards for public web sites are extremely important. Browsers need to support common standards so they can all be used with the widest range of web sites. And web applications need to support common standards so they do not alienate any segment of potential users.

- DataBase Management System (DBMS): almost all web applications require dynamic data. This data is typically stored in a database, a collection of information organized so that it can be easily accessed, managed, and updated. A DBMS plays the role to catalog, index, locate, retrieve, store data and maintain its integrity. A DBMS must address problems such as security, accuracy, consistency among different records, response time and memory requirements.

- Web Application Server: the core of a web application is the application server. The server handles all of the application logic and connectivity found in client-server applications. It works as a translator: it handles requests from the web browser, processes the business logic and builds the appropriate response.

Virtually all web applications have a common set of basic requirements, such as user session management (e.g., secure user login, password recovery), group management, authentication and authorization services, data persistence, templating systems and so on. A Web Application Framework usually includes all these functionalities, refined through hundreds of production deployments.

Web application framework are developed applying the following key design principles:

- Simplicity: less and simpler code should be written to use a framework. Avoid overuse of XML configuration files.

- Efficiency: applications should perform well and scale with support of clustering through sticky sessions preferred.

- Integration: a framework should not compete with good existing solutions, but should foster seamless integration.
• Reusability: component composition in a framework should be fully reusable and easy to distribute/deploy.

• Non-intrusive: HyperText Markup Language (HTML) or other markup should not be polluted with programming semantics, with compatibility with ordinary HTML editors and easy manipulation for graphics designer to recognize and avoid framework tagging.

• Diagnosis: when things go wrong, the framework should provide useful diagnostics and debugging information.

• Development tools: maximum tool support with minimum dependency on special tools.

Adopting Web Application Frameworks as the standard development infrastructure for web applications is the best way to ensure that development is not locked into any proprietary, dead-end architecture. This approach dramatically reduces technology churn and risk since the industry-standard open source frameworks are actively maintained and enhanced by highly skilled professionals worldwide. These developers take the responsibility of identifying the appropriate technologies, integrating the software, testing these technologies, and providing migration paths for existing users to the latest technology [Shan 06].

5.1.1 Mainstream design pattern of Web Application Frameworks

As we understood from the description of their architecture, WAFs tie up together different physic layers, quite often integrating different technologies. Therefore, unless structured approach is used in the design phase, Web Applications are prone to suffer of various problems which significantly hamper their extensibility and maintainability. This brought to the development of some standard design pattern focused on simplifying these problems. In the following we illustrate two widespread patterns also used by the framework that we introduce in the reminder of this chapter.

Model View Controller (MVC)

In the context of a Java 2 Enterprise Edition (J2EE) web application, the client will typically submit or request information to the server. The information is then either handed over to a Java Servlet that processes it, interacts
with a database and produces an HTML formatted response, or it is given to a Java Server Pages (JSP) document that combines HTML and Java code to achieve the same result. Both approaches are often considered inadequate for large projects because they mix application logic with presentation and make extension and maintenance difficult.

**MVC** is an architectural pattern used in software engineering. It separates the programming codes into three different areas, each of them specializes in one task, thus solving the above problem. MVC can accomplish J2EE application system’s stratification and the loose coupling of three layers or multilayers. It is a realization way of orienting dynamic content allowing us to manage the complexity of large software systems by dividing them into high-level components. The MVC mode identifies three core components: Model, View and Controller.

The **Model** is the main body of the Application, it stands for Business Data and Business Logic. It contains the application data and manages the core functionality.

The **View** represents the page design code, it renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes.

The **Controller** represents the navigational code, it receives user input and initiates a response by making calls on model objects. A controller accepts input from the user and instructs the model and viewport to perform actions based on that input.

### Dependency Injection and Inversion of Control (IOC)

In the development of a system using the object oriented programming methodology, we generally divide the system into objects where each of the objects represents some functionality. In this case, the objects in the system use some other objects to complete the given request. The objects with which our objects collaborate to provide the services are known as their dependencies. The traditional ways of obtaining the dependencies are by creating the dependencies or by pulling the dependencies using some factory classes or methods. But these approaches results in increasing both the complexity of the application and the development time-dependency. **Dependency Injection** is a design pattern that separates behavior from dependency resolution, thus decoupling highly dependent components. The fundamental principle of Dependency Injection is that application objects should not be responsible for looking up the resource or collaborators they depend on. Instead of hard-coding the dependencies, an injector (typically a container) should configure the objects, externalizing resource lookup from application code. Since this
process is fundamentally the inverse of normal practice, it is usually named Inversion of Control [Santosh 09, Arthur 05].

5.2 Web Application Frameworks used in the Vithea project

Going towards the direction of developing a highly modular, flexible, portable and re-usable prototype, Vithea has been built using the most widespread technologies available in terms of Web Application Frameworks. Therefore it adopts an integrated framework composed by Struts, Spring and Hibernate. In presentation tier Struts is used, in the business logic tier we have created business objects, business services and now we are going to integrate them using Spring framework, and Hibernate is used as object-relational mapper for the Data Access Layer. This compound framework provides the appropriate integration strategy for web application at all levels by combining the characteristics of these three frameworks and the layered idea of Web applications, which reduces the coupling degree in each layer of the system and makes the system easier to maintain and extend. Struts framework is a classical implementation of MVC architecture. Hibernate is a powerful technology for persisting data, and it enables Application to access data from any database in a platform-independent manner. Spring is a dependency injection framework that supports IOC. The beauty of Spring is that it can integrate well with most of the prevailing popular technologies, thus integrate Struts, Spring and Hibernate is a good pattern [Fang 08, Jiya 09].

5.2.1 Apache Struts 2

Apache Struts 2 is an open-source web application framework that has been developed in the Jakarta Project. It adheres to the MVC paradigm: the controller part is implemented by the StrutsPrepareAndExecuteFilter class, the model is represented by the Actions classes and the view by the Result. Figure 5.1 shows how MVC design pattern is achieved in the Struts 2 framework.

The controller is the first component to act in the processing, its job is to map requests to Actions. The role of the controller is played by the Struts 2 StrutsPrepareAndExecuteFilter. This element is simply a servlet filter that inspects each incoming request to determine which Struts 2 action should handle the request. The framework handles the entire controller work for us. We just need to inform the framework which request URLs map to which of
our actions. This can be done with XML-based configuration files or Java annotations.

The model represents the internal state of the application. Once the controller has determined which actions should handle the request it hands over control of the request processing to the action by invoking it. This invocation process, conducted by the framework, will both prepare the necessary data and execute the action’s business logic. When the action completes its work, it will be time to render a view back to the user who submitted the request. It will just forward the result to the Struts 2 view component.

The view is the presentation component of the MVC pattern. The result returns the page to the web browser. This page is the user interface that presents a representation of the application’s state to the user. These are commonly JSP pages, Velocity templates, or some other presentation-layer technology. While there are many choices for the view, the role of the view is clear-cut: it translates the state of the application into a visual presentation with which the user can interact.

![Struts 2 MVC Diagram]

**Figure 5.1: Struts 2 MVC**

Struts 2 is built from the ground up on proven best practices and community accepted design patterns. In addition Struts 2 introduces many architectural refinements. Thus, developing web applications with Struts 2 allows
us to benefit not only of the best practice, but also it provides some additional features designed to optimize both the customization of the workflow at different levels and the development in terms of presentation.

In order to further decrease coupling, request processing has been made modular by allowing a series of **interceptors** (custom or Struts 2 provided) to provide pre-processing and post-processing functionality. The underlying idea behind interceptors is that many Actions share common concerns, e.g: some Actions need input validated, other may need a file upload to be pre-processed and so on. Thus, Interceptors are Struts 2 components that execute both before and after the rest of the request processing. They provide an architectural component in which to define various workflow and cross-cutting tasks so that they can be easily reused as well as separated from other architectural concerns.

Moreover, the framework comes with a strong set of built-in **type converters** that support all common conversions. Data transfer and type conversion actually happen on both ends of the request processing cycle. The framework moves the data from the string-based HTTP requests to JavaBeans properties, which are clearly Java types. Moreover, the same thing happens on the other end. When the result is rendered, we typically convey some of the data from those JavaBeans properties back out into the resulting HTML page. Again, this means that the data has been reconverted from the Java type back out to a string format. The component who is delegated to carry out data between requests is the Action which actually implements JavaBeans properties for each piece of data that it wishes to carry. Instead, type conversion mechanism is carried out by OGNL technology. OGNL stands for the **Object-Graph Navigation Language** and has been integrated into the Struts 2 framework to help with data transfer and type conversion. OGNL is the glue between the framework’s string-based HTTP input and output and the Java-based internal processing.

The framework includes also powerful libraries and Plugin integration in order to provide a capable and flexible development of the front end side.

An extensive, improved **tag library** allows creating dynamic web applications drastically reducing the amount of code to be developed in the presentation layer, thus keeping it cleaner and easier for maintenance. Not only these tags contain output data but also provide style sheet driven markup that in turn helps in creating pages with less code. Here the tags also support **validation** and **localization** of coding that in turn offer more utilization. Furthermore, Struts 2 provides full support for advanced JavaScript and Ajax integration. AJAX is an acronym for Asynchronous JavaScript and XML. Essentially, a JavaScript can make an HTTP request and update portions
of a page directly, without going through a conventional POST or GET and refreshing the entire page. The key point is that when a script makes an Ajax request (XHR), the server does not know it came from a script, and handles it like any other request. It is not the Ajax request that is different, but the Ajax response. Instead of returning an entire page for the browser to display (or redisplay), an Ajax response will just return a portion of a page. The response can take the form of XML, or HTML, or plain text, another script, or whatever else the calling script may want.

Struts 2 also implements a validation framework which provides a versatile and maintainable solution for form field data validation. As with most of its components, validation has been a part of web application frameworks for a while, but Struts 2 takes it to a new level of refinement, modularity, and clean integration.

Last, another refinement of the framework is Internationalization. Struts 2 provides a clever way of making an application speak different languages for different users in an elegant fashion. The framework supports internationalization in the User Interface tags, in messages and errors coming from the Validation and in the Action classes [Brown 08].

The normal lifecycle of Struts 2, as shown in figure 5.2, begins with an initial
request sent from the client.

a) The request goes to the Servlet container which is passed through a standard filter chain.

b) Next, the required StrutsPrepareAndExecuteFilter is called, which in turn consults the ActionMapper to determine if the request should invoke an action.

c) If the ActionMapper determines that an Action should be invoked, the StrutsPrepareAndExecuteFilter delegates control to the ActionProxy. The ActionProxy consults the framework Configuration Files manager (initialized from the struts.xml file).

d) Next, the ActionProxy creates an ActionInvocation, which is responsible for the command pattern implementation. This includes invoking any Interceptors (the before clause) in advance of invoking the Action itself.

e) Once the Action returns, the ActionInvocation is responsible for looking up the proper result associated with the Action result code mapped in struts.xml.

f) Interceptors are executed again (in reverse order, calling the after clause). Finally, the response returns through the filters configured in the web.xml. If the ActionContextCleanUp filter is present, the StrutsPrepareAndExecuteFilter will not clean up the ThreadLocal ActionContext. If the ActionContextCleanUp filter is not present, the StrutsPrepareAndExecuteFilter will cleanup all ThreadLocals.

5.2.2 Hibernate and Java Persistence API

Persistence is one of the fundamental concepts in application development. When we talk about persistence in Java, we are normally talking about storing data in a relational database using Structured Query Language (SQL). A relational DBMS is a database based on the relational model developed by E.F. Codd [Wikipedia 11] in which all data and relationship among the data are represented in tabular form. A relational DBMS is not specific to a programming language, and a relational database is not specific to a particular application. Relational technology provides a way of sharing data among different applications or among different technologies that form part of the same application (the transactional engine and the reporting engine, for example). Relational technology is a common denominator of many disparate
systems and technology platforms. Hence, the relational data model is often the common enterprise-wide representation of business entities.

In an object-oriented application, persistence allows an object to outlive the process that created it. The state of the object may be stored to disk and an object with the same state re-created at some point in the future. This application is not limited to single objects - entire graphs of interconnected objects may be made persistent and later re-created in a new process. Most objects are not persistent; a transient object has a limited lifetime that is bounded by the life of the process that instantiated it. Almost all Java applications contain a mix of persistent and transient objects; hence we need a subsystem that manages our persistent data. Modern relational databases provide a structured representation of persistent data, enabling sorting, searching, and aggregation of data. DBMSs are responsible for managing concurrency and data integrity; they are responsible for sharing data between multiple users and multiple applications. A DBMS also provides data-level security. When we discuss persistence, we are thinking of storage, organization, and retrieval of structured data, concurrency and data integrity and data sharing.

In particular, we are thinking of these problems in the context of an object-oriented application that uses a domain model. An application with a domain model does not work directly with the tabular representation of the business entities; the application has its own, object-oriented model of the business entities. Then, instead of directly working with the rows and columns of an SQL result set, the business logic interacts with this object-oriented domain model and its runtime realization as a graph of interconnected objects. The business logic is never executed in the database (as an SQL stored procedure), it is implemented in Java. This allows business logic to make use of sophisticated object-oriented concepts such as inheritance and polymorphism.

An object model uses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency, polymorphism and persistence. The object model enables you to create well-structured, complex systems. In an object model system, objects are the components of the system. Objects are instances of classes, and classes are related to other classes via inheritance relationships. An object has an identity, a state and behavior. An object model helps you create reusable application frameworks and systems that can evolve over time. See figure 5.3 for a visual representation of an object model related to a domain which considers a Book and its Publisher.

A relational model defines the structure of data, data manipulation and data integrity. Data is organized in the form of tables, and different tables are associated by means of referential integrity (a foreign key). Integrity
constraints such as a primary key, unique check constraints, and not null
are used to maintain an entity’s integrity in the relational model. Figure
5.4 shows a visual representation of a relational model also related to the
domain of a Book and its Publisher. A quick comparison between the two
figures allow us to show the main differences: the object model is limited to
representation of a properties with the description of its type - bookId:long -,
while the relational model in the description comprises integrity constraints
too - bookid bigint(19) not null (pk) .

A relational data model is not focused on supporting entity-type inher-
itance: entity-based polymorphic association from an object model can not
be translated into similar entities in a relational model. In a object model,
you use the state of the model to define equality between objects. But in a
relational model, you use an entity’s primary key to define equality of en-
tities. Object references are used to associate different objects in an object
model, whereas a foreign key is used to establish associations in a relational
model. Object references in the object model facilitate easier navigation
through the object graph. Because these two models are distinctly different,
it is needed a way to persist object entities (Java objects) into a relational
database [Mak 10].

In other words, the heart of the problem is translating the logical repre-
sentation of the objects into an atomized form that is capable of being stored
on the database, while somehow preserving the properties of the objects and
their relationships so that they can be reloaded as an object when needed. If
this storage and retrieval functionality is implemented, the objects are then
said to be persistent.
Object-Relational Mapping (ORM)

Object-relational mapping frameworks help to take advantages of the features present in the object model and the relational model. With the help of ORM frameworks, we can persist objects in Java to relational tables using metadata that describes the mapping between the objects and the database. The metadata shields the complexity of dealing directly with SQL and helps developing solutions in terms of business objects.

In ORMs frameworks an entity is a lightweight persistence domain object. Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The primary programming artifact of an entity is the entity class, although entities can use helper classes. The persistent state of an entity is represented either through persistent fields or persistent properties. These fields or properties use object-relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.

ORMs framework provide an object-oriented query language, so that operations are conveniently performed in the manner of a class of objects. When working with an SQL database in a Java application, the Java code triggers SQL statements to the database via the JDBC API. The SQL itself might have been written by hand and embedded in the Java code, or it might have been generated on the fly by Java code. We use the JDBC API to manually handle database connections, bind arguments to query parameters, initiate execution of the query, scroll through the query result table, retrieve values from the result set, manually handle associations and so on. These are low level data access tasks, SQL statement are not hard, they are just tedious and error prone [Bauer 05].

For demonstration purpose, still using previous example domain, we show a portion of the code needed to make query with plain JDBC API's.
public List getBook() throws CustomException {
    Connection c = null;
    PreparedStatement p = null;
    List list = new ArrayList();
    final String JDBC_DRIVER = com.mysql.jdbc.Driver;
    final String DB_URL = jdbc:mysql://localhost/BookShopDb;
    final String USER = username;
    final String PASS = password;

    try {
        Class.forName(com.mysql.jdbc.Driver);
        c = DriverManager.getConnection(DB_URL, USER, PASS);
        p = c.prepareStatement("SELECT * FROM BOOK WHERE ISBN = ?");
        p.setString(1, 1234567890);
        ResultSet rs = p.executeQuery();
        while (rs.next()) {
            String text = rs.getString(1);
            list.add(text);
        }
        rs.close();
        p.close();
        c.close();
        return list;
    }
    catch (Exception e) {
        throw new CustomException(
            Failed to retrieve books from db., e);
    }
    finally {
        if (p != null) {
            try {
                p.close();
            }
            catch (SQLException e) {
                log.log(Level.WARNING,
                    Could not close open statement.);
            }
        }
        if (c != null) {
            try {
                c.close();
            }
            catch (SQLException e) {
                log.log(Level.WARNING,
                    Could not close open connection.);
            }
        }
    }
}
For the sake of completeness we show the same code using Hibernate

```java
public List<Book> readAll(String isbnCode) {
    Session session = SessionManager.getSessionFactory().
    .getCurrentSession();
    Query query = session.createQuery
    ( select b from Book b where b.isbn = ? );
    query.setString(0, isbnCode);
    return query.list();
}
```

We are indeed more interested in the business problem that requires this data access, not at the manner this is performed.

Thus, using an ORM framework allows to improve productivity since we use metadata to persist and since we do not have to concern ourselves of writing complex SQL statements we could be more concentrate on the business logic. Furthermore because much of the work is done through configuration, our code has fewer lines requiring so less maintenance, makes system more understandable and easier to refactor. Last, abstracting an application from the underlying SQL database and SQL dialect, the portability to support multiple databases is achieved by few changes in the configuration file.

**Hibernate**

Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions. This lets users to develop persistent classes following object-oriented principles such as association, inheritance, polymorphism, composition, and collections.

Hibernate is an ambitious project that aims to be a complete solution to the problem of managing persistent data in Java. It mediates the application’s interaction with a relational database, leaving the developer free to concentrate on the business problem at hand. Hibernate is an non-intrusive solution. It means that there is any requirement to follow many Hibernate-specific rules and design patterns when writing our business logic and persistent classes; thus, Hibernate integrates smoothly with most new and existing applications and does not require disruptive changes to the rest of the application. Figure 5.5 shows how Hibernate fits into an application between the client code and the database.
Besides the advantages already illustrated of an ORM, Hibernate implements some important performance enhancements. Using Hibernate we will get benefit of Cache, in fact Hibernate support two level of cache, first level and 2nd level, so we can store our data into Cache for better performance. Also, Hibernate does not need to create any connection pool for managing and reusing connection with the database, we can use c3p0 [C3P0 11], an open source JDBC connection pool distributed along with Hibernate. Furthermore, Hibernate allows lazy loading. Let us think on a tree data structure, we have a node parent with a collection of node children. Lazy-load means that when loading the parent Hibernate does not actually load all the children. Instead, it loads them when requested to do so. Last, we do not need Query tuning. If we use Criteria Queries in Hibernate then it automatically tuned our query and return best result with performance.

**Java Persistence Api (JPA)**

The purpose of Hibernate is to allow to treat a database as if it stores Java objects. However, databases in practice do not store objects, they store data in tables and columns. As we have exposed, for achieving this Hibernate needs something to map which tables relate to which objects. This informa-
tion is usually provided in an XML mapping file. Mappings between entities and objects can be also accomplished using Annotations. An annotation, in the Java computer programming language, is a special form of syntactic metadata that can be added to Java source code. Hibernate supports object-relational mapping metadata through annotations implementing JPA standards. JPA, Java Persistence API framework came with the release 5.0 of J2EE.

There are powerful benefits of replacing Annotation with Xml mapping file: annotations-based mappings are far more intuitive than their XML-based alternatives, as they are written in the source code along with the properties that they are associated with. Also, annotations are less verbose and more transparent than their XML equivalents. Since the annotations are compiled directly into the appropriate class files, there is less risk of a missing or stale mapping file causing problems at deployment.

Finally, Hibernate annotations are not considered a standard, while its underlying implemented solution, JPA, it is. Indeed an entity mapped exclusively by using JPA annotations can be supported by other JPA-compliant ORM tools, while Hibernate specific Annotation are specific to Hibernate. Following excerpt of code -listing 5.3- shows how to persist a Java Object with JPA annotations.

```
package com.example.domain;

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.Id;

@Entity
@Table(name = BOOK)
public class Book {
    @Column (name = isbn )
    @Id
    String isbn ;

    @Column (name = book_name )
    String bookName ;

    @Column (name = publisher_code )
    String publisherCode ;

    @Column (name = publisher_date )
    Date publishDate ;
```
```java
@Column (name = "price")
Long price;

/**
 * @return the isbn
 */
public String getIsbn() {
    return isbn;
}

/**
 * @param isbn the isbn to set
 */
public void setIsbn(String isbn) {
    this.isbn = isbn;
}

/**
 * Other getter and setter methods
 ............
 ............
 */
```

5.2.3 Spring

The Java language and platform provides a wealth of functionality for architecting and building applications, ranging all the way from the very basic building blocks of primitive types and classes, to rich full-featured application servers and web frameworks. However, the problem of taking the basic building blocks and composing them into a coherent whole has been typically left to the architects and developers who are tasked with building an application.

Spring framework may aid in fulfilling this task addressing most infrastructure concerns of enterprise applications. Spring framework provides a number of services, it implements several commonly used best practices and makes these objects available to integrate with our projects instead of coding these patterns as a part of our project. As a general implementation guideline in web application designs, Spring may be considered as a preferred baseline, serving as the backbone for the business object layer, or middle tier, of a J2EE web application.

Spring is a multi-tier open-source lightweight application framework organized in a layered architecture consisting of twenty well-defined modules.
These modules are grouped into Core Container, Data Access/Integration, Web, Apsect Oriented Programming (AOP), Instrumentation, and Test, as shown in figure 5.6. A brief introduction of the main building block is given [Johnson 10]:

- The main components of the Core Container consists of the Core, Beans, Context, and Expression Language modules. The Core and Beans modules provide the fundamental parts of the framework, including the IOC and Dependency Injection features. The Context module adds support for internationalization, event-propagation, resource-loading, and the transparent creation of contexts.

- The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules. The JDBC module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding. The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis. The OXM module provides an abstraction layer that supports Object/XML mapping implementations. The Java Messaging Service (JMS) module contains features for producing and consuming messages. The Transaction module supports programmatic and declarative transaction management.

- The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules. Spring’s Web module provides basic web-oriented integration features. The Web-Servlet module contains Spring’s model-view-controller (MVC) implementation for web applications. The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application. The Web-Portlet module provides the MVC implementation to be used in a portlet environment.

- Spring’s AOP module provides an AOP Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors to cleanly decouple code. The separate Aspects module provides integration with AspectJ. The Instrumentation module provides class instrumentation support and classloader implementations to be used in certain application servers.

- The Test module supports the testing of Spring components with JUnit or TestNG.

However, Spring is designed in a modular fashion, and does not force to use necessarily these modules altogether. In fact, each of the modules
can stand on its own or be jointly combined with a subset of the whole set of components in the framework. In the following we will not linger in describing the details of the whole framework, instead we will only deepen a specific component of our interest, namely the Core Container.

The core container is a lightweight IOC container which defines how the objects (here referred as Beans) of our applications are instantiated, assembled and managed.

By using the model of IOC the implementation of the object becomes transparent, the coupling between the components business is reduced and the system compatibility and scalability get improved. The container achieve this, by injecting object’s dependencies when it creates the bean, controlling the instantiation or location of its dependencies by using direct construction of classes, or a mechanism such as the Service Locator pattern.
The \texttt{org.springframework.beans} and \texttt{org.springframework.context} packages are the basis for Spring Framework’s IOC container. The \texttt{BeanFactory} interface is actually the responsible for allowing to decouple application’s configuration and dependency specification from the actual application code. It provides an advanced configuration mechanism capable of managing any type of object. It is responsible for loading bean definitions stored in a configuration source (such as an XML document), and use the \texttt{org.springframework.beans} package to configure the beans.

Below a simple example of a bean definition is provided. These bean definitions correspond to the actual objects that make up an application. Typically here are defined service layer objects, Data Access Objects (DAOs), presentation objects such as Struts Action instances, infrastructure objects such as Hibernate SessionFactories, JMS Queues, and so forth.

<table>
<thead>
<tr>
<th>Listing 5.4: Excerpt of a Spring configuration file</th>
</tr>
</thead>
</table>

```xml
  <bean id="anotherExampleBean" class="examples.AnotherBean"/>
  <bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
  <bean id="exampleBean" class="examples.ExampleBean">
    <property name="beanOne">
      <ref bean="anotherExampleBean"/>
    </property>
  </bean>
  <property name="beanTwo" ref="yetAnotherBean"/>
  <property name="integerProperty" value="1"/>
</beans>
```

The \texttt{id} attribute is a string used to identify the individual bean definition. The \texttt{class} attribute defines the type of the bean and uses the fully qualified classname. The value of the \texttt{id} attribute refers to collaborating objects.

In conclusion the use of a lightweight container such as the Spring Framework does have definite advantages. Because of its a layered architecture it is complete and is modular which allows to incrementally adopt the framework into a project. Even choosing to use just about any part of it in isolation, yet its architecture is internally consistent. Spring framework is non-invasive due to the IOC pattern, eliminating the creation of singletons and factory
classes thus providing low coupling dependencies. Spring can facilitate good programming practice by reducing the cost of programming to interfaces, rather than classes, almost to zero.
Chapter 6

Vithea System

In this chapter we detail how the Vithea system has been built, describing both pedagogical as well as technical choices made while building the system. Two different modules have been developed: the Administration Module which provides support for the clinicians and the Client Module, specially designed for aphasic individuals. These are described in section 6.1. Then, in section 6.2 we deepen the integrated approach of the three aforementioned Web Application Frameworks (Struts 2, Hibernate, Spring), going into the details of the project’s architecture.

6.1 Administration and Client Module

The Administration module has been specially designed for clinicians: it allows the creation of exercises as well as monitoring the user’s performance in terms of frequency of access at the system and user’s progress. On the other hand, the Client module is meant to be used by the patients to perform the therapeutical exercises. It has been implemented by the integration with a Flash application, allowing to support rich multi-media interactions (including audio recording/playing) from a standard Web browser. Particularly, users are aphasic patients, people who typically are victim of a cerebral vascular accident and in some cases have also reported some forms of physical disabilities. For this reason, special care has been taken in designing the client interface.

Although the two applications have been specialized for different classes of users and therefore present different functionality, they share the key purpose of the project: the training’s exercises. As we saw in section 2.1, difficulties in recalling word or names is the most common disorder presented in aphasia. Disorders like naming abilities in the literature are treated with se-
mantic exercises, such as Naming of object, Naming common actions, Picture matching and other more [Adlam 06]. In the Vithea system the particular class of training exercises reflect the same guidelines. The set of therapeutic exercises integrated in the Vithea System have been designed by the the Language Research Laboratory of the Department of Clinical Neuroscience of the Lisbon Faculty of Medicine (LEL). LEL has provided a rich battery of exercises, these are classified into two macro-categories that represent the type of stimulus, namely:

A) Visual

- Naming object picture
- Naming of verbs with videos
- Naming of verbs with object’s images
- Phonological Evocation
- Semantics Evocation

B) Visual, Oral or Visual and Oral

- Responsive Naming
- Complete Sayings
- Part-whole Associations
- What name is given to...
- Generic Designation
- Naming by function

All the above exercises are based on a common scheme. The patient is provided with a textual, visual (pictorial or video) or audio stimulus, and is asked to respond to it orally (e.g. by describing the image he sees or completing a popular saying). Even though the system has been designed to be flexible and extensible enough to support the whole set of the above mentioned exercises, the focus of this work (especially for what concerns its evaluation) has been on Naming object picture exercises.

In particular, in the Vithea system, we have integrated a set of Naming object picture exercises that extend and adapt the corpus of line drawings by Snodgrass and Vanderwart [Snodgrass 80] to the Portuguese languages. The original pictures are black-and-white line drawings executed according to a set of rules that provide consistency of pictorial representation, figure 6.1
shows some images from the corpus. The concepts were selected to provide exemplars from the following several widely studied semantic categories:


The patient is shown one of these pictures and is asked to name the object it depicts. The user replies verbally to the quiz and its utterance is recorded, encoded and sent via network to the server side. Here, a web/application server receives the audio file via a servlet that serves as an interface to the ASR system. This takes as input the audio file encoding the patient’s utterance and generates a textual representation of it. This is then compared with a set of predetermined textual answers (for that given question, of course) in order to verify the correctness of the patient’s input. Finally, a feedback is sent back to the user, who, in case of failure, can retry up to a user-specifiable number of times. A comprehensive view of this architecture is shown in figure 6.2.

In addition to the set of training exercises, which are meant to be used on a daily basis by the aphasic patient, the Vithea system supports also a different class of exercises: **Evaluation Exercises**. Unlike training exercises, evaluation exercises are used by human therapists to periodically assess the patients’ progress and his/her current degree of aphasia via an objective metric denoted as QA.
Figure 6.2: Architectural Overview of the Vithea System.

Evaluation exercises are chosen from a subset of the previously mentioned classes of therapeutic exercises, namely:

- *Naming object picture*
- *Naming of verbs with object’s images*
- *Verb Generation*

### 6.1.1 Administration Module

The Administration Module plays the role to allow clinicians to build the training sessions. The module is composed by three submodules: **User, Exercise, Statistic**.

**User sub module** allows to handle a knowledge base of patients. Besides basic information related to the user personal profile, the database also stores for each individual his/her type of aphasia, his/her aphasia level and a QA information. Aphasia level is a subjective value comprised between 0 and 6, while QA is an objective measure of aphasia’s severity and is based on the result value accomplished in three evaluation exercises.
**Exercise sub module** allows to create, update, preview and delete stimuli from an exercise. An exercise is composed by a varying number of stimuli. In addition to the canonical valid answer, the system accepts for each stimuli an extended word list comprising three extra valid answers, this list allows to consider the most frequent synonyms and diminutives.

This sub-module does not only allow the management of the set of stimuli. It also provides a rich Web based interface to manage the database of multimedia resources used to define new stimuli (or update existing ones). The system is capable to handle a wide range of multimedia encoding:

- audios (accepted file types: *wav, mp3*),
- videos (accepted file types: *wmv, avi, mov, mp4, mpe, mpeg, mpg, swf*),
- images (accepted file types: *jpe, jpeg, jpg, png, gif, bmp, tif, tiff*).

In order to maximize performance, it has been chosen to store only the resource’s metadata in a database management system. On the other hand, resources are stored directly into the file system. At creation time, a reference to the path of the media file is stored in the database as well as some additional meta data (such as file title, additional comments), in order to allow a fast recognition at utilization time. A preview feature for media files has also been provided. Given the diversity of the various files typologies accepted by the system, a conversion to a unique file type was needed, in order to show them all with only one external tool. Audio files are therefore converted to *mp3* file format, while video files are converted to *flv* file format.

The file system is essentially structured following file typologies into three main directories: Audio, Video and Images. Some sub directories are created under Audio and Video folders in order to store the converted version of the same file. A tree structure is created under Images folder, representing the logical classification of themes for the *Naming object picture* exercise.

**Statistics sub module** allows to monitor both the frequency with which users access the system and users progress done within the execution of the training exercises. We thought that monitoring the utilization of the application from the users could be an important feedback about system’s feasibility. This is motivated by a common concerns about the fact that some users have abandoned their therapeutics session when he/she was not able to see quick results in terms of improvements.
6.1.2 Client Module

The Client Module plays the role to train aphasic individuals. It is also in charge of storing statistical data about user sessions. As we have already mentioned, particular attention has been given to the design of the graphical user interface (GUI) for this module. We acknowledge the eventuality that the system could be used by people with some physical impairment like for instance a reduced arm mobility due to CVA and as a direct consequence they may experience problems using a mouse. Moreover, even though aphasia is increasing in the youngest age groups, it still remains a predominant disorder among elderly people. These groups of people are more likely to suffer of visual deficit. For these reasons the user interface must be as simple and clear as possible, yet appealing at the same time. Thus, we reduced the number of hyperlinks to a minimum in order to avoid as much as possible the use of mouse and we used big icons for representing our interface elements.

Figure 6.3: A capture image of the Client module interface that lists the preview of an exercise’ questions

Once logged in, the system shows an elementary menu, which lists all the exercises divided by the macro-categories aforementioned. Following the
advice provided by the staff from the Language Research Laboratory, after
choosing an exercise the system first shows a graphical preview of all the
stimuli contained in that exercise. Then, the user can eventually start his/her
training session by selecting the corresponding option or go back to the main
list for picking another exercise. Figure 6.3 illustrates this functionality.

This alternative is motivated by a pedagogical reason: leaving the user
free to explore the system without forcing he/she to start a specific training
session as soon as is selected, helps to let he/she feels comfortable with the
program. Once the user has chosen to start a training exercise, the system
will present target stimuli one at a time in a random way. The user should
record his/her answer, which is immediately sent to the ASR system for
processing. The processed utterance is then compared to the desired correct
response and the result from the comparison is provided back to the user.
As mentioned before it is possible to record again the processed utterance
up to a number of time chosen before starting the exercise.

Besides permitting training sessions, the Client Module has the respon-
sibility of storing user’s utterances, each user access at the system and
start/end time of a training exercise and of each stimulus. Our purpose
is to log every single access in order to evaluate the impact and effectiveness
of the program by seeing the frequency with which it is used. While we
record the total time needed to accomplish a single stimuli or to end a whole
exercise in order to estimate user performance improvements.

\section{6.2 Architectural Overview}

As we have partially introduced when we talked about Web Application
Frameworks in chapter 5, for the development of the Vithea system we have
chosen to adopt an integrated framework composed by Struts 2, Hibernate
and Spring. Combining the characteristics of these three frameworks we are
able to reduce the coupling degree in each layer of the system, making main-
tainability and extensibility easier. Struts 2 focuses on reducing the coupling
between performance and logic, while devoting less attention to the rele-
vance between the business layer and the persistence layer. Hibernate acts
as a bridge between Java applications and the relational database, taking
care of the mapping between Java objects and data provided in the form of
set of rows. Spring acts as a glue between all the layers, by injecting objects
dependencies makes their implementation transparent. The Spring compo-
nent which defines how dependencies between objects are created, configured,
and managed is the Core Container and it is actually the only component of
the framework used in the project.
In the following we deepen how these three frameworks tie up all together, illustrating more closely both the role taken from each of them and how each one effectively carries out its own part. To this purpose, figure 6.4 simulates the whole cycle of a request coming from the browser up to its processing and its consequently response from the application server, highlighting for each step which framework comes into action.

![Diagram of the Vithea system](image)

1. **Presentation Tier** both the Administration and the Client modules interact with the server either for updating or only for consulting data by sending information through the browser. The JSPs pages in presentation tier collect these data and connect with the middle tier by the controller, then the middle tier will connect with the data tier. At last, the data returned from the middle tier will be displayed again by the JSP pages using Struts Taglibs, Tiles, Ajax, ActionScript®.

2. **Middleware Tier** this tier can be further divided into four parts as follows:
a) **Web tier:** the Controller of Struts is responsible for the data exchange between the presentation tier and the business logic tier, while the Spring Ioc container plays its role by injecting actions, business objects and their dependencies.

b) **Service (business logic) tier:** besides exchanging information between Web tier and DAO objects this tier also implements the actual business logic of the application by providing data with extra information or functionality.

c) **DAO (Data Access Object) tier:** encapsulates data management functionality and handles information exchange between the Service tier and the PO tier.

d) **PO (Persistence Object) tier:** relational database tables which store application information are mapped to objects by the object/relational mapping tool Hibernate, thus database tables could be handled as objects.

3. **Data Tier** Here Database Server actually stores and retrieves information. By providing data its own tier keeps data neutral and independent from application servers or business logic.

### 6.2.1 Presentation Tier

The presentation tier in the Vithea system has been built with the support of different technologies. Besides Struts 2 tag library and AJAX, which both came along with the framework, the presentation tier also relies on Apache Tiles templating framework [Tiles 11] and Adobe® Flash® technologies. Tiles is used for the construction of every single JSP page of the project, achieving in this way modularity and re-utilization also in the presentation tier. Adobe® Flash® is used for developing a special module of the application, the one that allows users recording and evaluating their utterance.

**Apache Tiles**

Tiles is a templating framework designed to easily allow the creation of web application pages with a consistent look and feel. It can be used for both page decorating and componentization. Tiles allows to define page fragments that can be assembled into a complete page at runtime. These fragments, or tiles, can be used as simple includes in order to reduce the duplication of common page elements or embedded within other tiles to develop a series of reusable templates. These templates streamline the development of a consistent look and feel across an entire application.
Listing 6.1 shows how a page-template is defined by using several different JSPs, which in turn play their own role inside the page. Many layouts may be defined and a new page may extends any of them, improving appropriate redefinitions for customization where necessary.

Listing 6.1: Excerpt of a Tiles configuration file

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles−definitions PUBLIC
"−//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
"http://tiles.apache.org/dtds/tiles−config_2.0.dtd">
<tiles−definitions>
<definition name="mainLayout" template="/jsp/common/layouts/vithea−layout.jsp">
<put−attribute name="title" value="Vithea Administration Module"/>
<put−attribute name="header" value="/jsp/common/layouts/header.jsp"/>
<put−attribute name="bodyTitle" value=""/>
<put−attribute name="body" value="/jsp/common/layouts/body.jsp"/>
<put−attribute name="errors" value="/jsp/common/layouts/errorMessages.jsp"/>
<put−attribute name="menu" value="/jsp/common/layouts/menu.jsp"/>
<put−attribute name="footer" value="/jsp/common/layouts/footer.jsp"/>
</definition>
<definition extends="mainLayout" name="user.create" >
<put−attribute name="bodyTitle" value="Inserir novo utilizador"/>
<put−attribute name="body" value="/jsp/adminUser/createUser.jsp"/>
</definition>
</tiles−definitions>
```

Listing 6.2 shows how a template is constructed using the mechanism provided by Tiles. Lines 4 and 5 include the library specifying that it should be referred with the prefix tiles. Lines 21, 27, 32, 37 indicate the sections whose content will be replaced with the values of the attributes used in the template definition (listing 6.1 - lines 11-14, 16-17, 22-23).
Listing 6.2: Excerpt of Vithea’s main template

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<%@ taglib prefix="sx" uri="/struts-dojo-tags" %>
<!DOCTYPE html PUBLIC "//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <link rel="stylesheet" type="text/css" href="<s:url value="jsp/css/layout.css" />" media="screen"/>
  <script language="JavaScript" type="text/javascript"
       src="<s:url value="/jsp/js/swfobject.js"/>
       charset="utf-8"/>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  <title><tiles:getAsString name="title"/></title>
</head>
<body>
  <div class="header">
    <tiles:insertAttribute name="header" />
  </div>
  ........
  ........
  <div class="mainWithMenu">
    <tiles:insertAttribute name="body" />
  </div>
  ........
  ........
  <div class="footer">
    <tiles:insertAttribute name="footer" />
  </div>
</body>
</html>
```

Listing 6.3 shows how a result coming from the processing of an Action is automatically formatted via the tiles mechanism by the means of the property `type="tiles"` inside the `result` child of the `action` tag (lines 14, 15). The value `user.create` of the `result` tag is the same as the one expressed in the tiles configuration file (listing 6.1 - lines 27-32).
Flash based Audio Recorder

The client application is the one delegated for the training session. For recording users’ utterances an external module has been integrated within the system. To this purpose we have evaluated various different technologies, namely:

- Java Applet [Java 10],
- Macromedia Flash [Platform 11],
- Audio Recorder, a custom ready-to-use module developed by the School of Education at the University of Nottingham released under GNU license [of Nottingham 09].

An applet is a program written in the Java programming language that can be included in an HTML page much in the same way an image is included in a page. Unfortunately this technology first requires a Java-enabled browser in order to correctly display a page that contains an applet. Second, when an applet is loaded its code is transferred to the client system and executed by the browser’s Java Virtual Machine (JVM). This, in turns, means that an applet which executes on a client machine and subsequently tries to send data to a remote web application needs to be signed. The signature is a way to verify that the applet or application is from a reliable source and can be trusted to run in the client machine.
Macromedia Flash is a multimedia platform used to add animation, video, and interactivity to web pages. Flash manipulates vector and raster graphics to provide animation of text, drawings, and still images. It supports bidirectional streaming of audio and video, and it can capture user input via mouse, keyboard, microphone, and camera. Flash contains an object-oriented language called ActionScript\textsuperscript{®} [Actionscript 11]. Flash content may be displayed on various computer systems and devices, using Adobe\textsuperscript{®}Flash\textsuperscript{®} [Player 11], a cross-platform browser-based application runtime available free of charge for common web browsers.

Audio Recorder provided by the Notthingham University is a "learning tool that allows students and teachers to record an MP3 audio file directly from a web browser and save it to a local folder". It was developed to support online foreign language learners to practice pronunciation, therefore apparently seemed to fit perfectly our needs. Unfortunately it presents two main disadvantages: it only works in Internet Explorer browsers and also needs to be signed since it uses an ActiveX control in order to convert audio to MP3.

Therefore, after these evaluations we decided that Flash technology, among the others, is the one that more easily integrates in most browsers without any other kind of effort in terms of installing extra plugin or security’s issue. Given the particular kind of users who will use the system, these observations have been considered as first priority with respects to any other requirements.

Thus, we took the advantages offered from the Adobe\textsuperscript{®}Flash\textsuperscript{®} and we developed an external component that provides the specific functionality for our needs. The interface has been designed following the usability requirements we have already considered for the overall application. We used a reduced set of large symbols and we tried to lessen at the bare minimum the number of interactions required. Recording and sending an utterance to the server involves only two actions to take, while another one is required to play back the audio recorded.

Figure 6.5 provides a comprehensive vision of the Audio Recorder. Figure 6.5 - a) illustrates the interface before the recording phase, figure 6.5 b) illustrates the interface during the recording phase, last 6.5 c) illustrates the interface once an utterance has just been recorded and is available for being listened again.

The code of the component has been written using ActionScript\textsuperscript{®}3.0 programming language, while the communication between it and the surrounding JSP page has been achieved whit Javascript language. Basically, when a user starts recording, the default microphone device available is accessed and a \textit{RecordingEvent} begins to store data in memory as an array of bytes.
When recording stops an `Event.COMPLETE` is dispatched allowing data to be encoded in WAV format and then sent to the server. After the ASR processing, a feedback is sent back to the user who is now able to play back the data recorded.

### 6.2.2 Middleware Tier

Besides libraries and business logic, the middleware tier in the Vithea system includes Apache Tomcat [Tomcat 11] an open source servlet container developed by the Apache Software Foundation (ASF) [Apache 11]. Tomcat is released under Apache License version 2 [License 11], version used is 6.0.24-5.10.1.

Also, the application relies on some additional tool needed for converting
and working with the various media files accepted from the system. All video files are converted to a common format, flv. This is achieved through a Video Encoder Engine for Adobe Flash (FVEC) [SourceTec 10]. Instead, FFmpeg [FFmpeg 10], a cross-platform solution to record, convert and stream audio and video, has been used both for allowing conversion from wav to mp3 file format and for extracting a caption image of the first second of the videos. This was needed in order to show a small preview when videos are listed in a page.

We have already described the framework used in the project in chapter 5, now our intention is detailing how these work together. However before going in depth through the heart of our project, we need to show with the help of figure 6.6 how a standard web application is structured when deployed in a Web Application Server. This is needed for a better understanding of our following explanation. File web.xml that resides under the directory WEB-INF is the entry point of the application, the servlet container looks for it to determine what classes are needed to activate the Web application.

**Figure 6.6: Web Application structure**

**Configuring Spring**

In order to make the Spring framework aware that it is in charge of all the applications dependencies, two steps need to be followed: of course downloading the necessary library and configuring the container for loading the
org.springframework.web.context.ContextLoaderListener class. This is done by registering in the web.xml a listener element and its listener-class child. Then, a context parameter called contextConfigLocation determines the location of the Spring configuration file. The context parameter is configured using the context-param element, which has two children that specify the parameter name and its value.

Listing 6.4: Excerpt of Vithea’s web.xml configuration file

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<web-app
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
<display-name>Vithea Administration Module</display-name>

<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

<context-param>
  <param-name>
    contextConfigLocation
  </param-name>
  <param-value>
    /WEB-INF/classes/applicationContext.xml
  </param-value>
</context-param>

......
......

</web-app>
```

The XML in listing 6.4 does the following:

1. Specifies that the servlet container should load the Spring context listener

2. Specifies that the Spring context listener will load the application’s Spring configuration from /WEB-INF/classes/applicationContext.xml
The `applicationContext.xml` file registers all the Spring Beans - a bean is an object or class instance that is created and managed by the container - needed for the application to work. These may include, among many others that depend from specific needs of the application, configuration for database connection, transaction configuration, security configuration as well as customized business service configuration. Listing 6.5 shows a fragment of this file that registers the Bean to which is delegated the responsibility to login a user into the application.

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                      http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
  <default-lazy-init="true">
    ......<br />
    ......<br />
  <bean id="loginUserBO"
        class="pt.inesc.l2f.vithea.bo.impl.LoginUserBoImpl">
    <property name="userDao" ref="userDao"/>
  </bean>
    ......<br />
    ......<br />
  <import resource="pt/inesc/l2f/vithea/conf/user/spring-user.xml"/>
  <import resource="pt/inesc/l2f/vithea/conf/exercise/spring-exercise.xml"/>
    ......<br />
    ......<br />
</beans>
```

Class `pt.inesc.l2f.vithea.bo.impl.LoginUserBoImpl` encapsulates the business logic for the login action. The property `userDao` (listing 6.6, line 10) actually is an example of Dependency Injection, it refers to the class `pt.inesc.l2f.vithea.dao.impl.UserDaoImpl` whose bean declaration is registered into the file `spring-user.xml` together with other beans related to User management. This file, among others, is imported as external resources for a better readability and maintainability (lines 14-17). Listings 6.6 illustrates how these classes are related via Dependency Injection, which is realized through `userDao` and the methods `getUserDao()`, `setUserDao(…)

```java
package pt.inesc.l2f.vithea.bo.impl;

import pt.inesc.l2f.vithea.bo.LoginUserBo;
```

Listing 6.6: LoginUserBoImpl class
```java
import pt.inesc.i2f.vithea.dao.impl.UserDaoImpl;
import pt.inesc.i2f.vithea.entity.User;

public class LoginUserBoImpl implements LoginUserBo {

    private UserDaoImpl userDao;

    public boolean checkUserData(String user, String password) {
        User usr = getUserDao().getUser(user);
        if (usr == null) {
            //add error user name was wrong or user does not exists
            return false;
        } else {
            if (usr.getPassword().equals(password))
                return true;
            else {
                //add error password was wrong
                return false;
            }
        }
    }

    public UserDaoImpl getUserDao() {
        return userDao;
    }

    public void setUserDao(UserDaoImpl userDao) {
        this.userDao = userDao;
    }

    public User getUser(String userName) {
        return getUserDao().getUserByUserName(userName);
    }
}

Integrating Spring with Struts 2

We have already described the normal flow of a request handled by Struts 2 framework. StrutsPrepareAndExecuteFilter is the one that is actually delegated to monitoring request and response from client to the servlet, it invokes the ActionMapper to determine which Action should be invoked. Thus, for
```
completeness listing 6.7 shows another excerpt of the web.xml that configures Struts 2 StrutsPrepareAndExecuteFilter

<table>
<thead>
<tr>
<th>Listing 6.7: Excerpt of Vithea’s web.xml configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;filter&gt; &lt;filter-name&gt;struts2&lt;/filter-name&gt;</td>
</tr>
<tr>
<td>&lt;filter-class&gt;</td>
</tr>
<tr>
<td>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</td>
</tr>
<tr>
<td>&lt;/filter-class&gt;</td>
</tr>
<tr>
<td>&lt;init-param&gt;</td>
</tr>
<tr>
<td>&lt;param-name&gt;actionPackages&lt;/param-name&gt;</td>
</tr>
<tr>
<td>&lt;param-value&gt;pt.inesc.12f.vithea.action&lt;/param-value&gt;</td>
</tr>
<tr>
<td>&lt;/init-param&gt;</td>
</tr>
<tr>
<td>&lt;/filter&gt;</td>
</tr>
</tbody>
</table>

However, we have not yet highlighted that the responsible for building the core framework objects when using exclusively Struts 2 framework is the ObjectFactory class. In order to enable the integration between these two frameworks, one needs to download and include Spring plugin library. Spring plugin works by overriding the Struts ObjectFactory (listing 6.8 lines 8,9), this will enable Spring to inject into the Action classes any dependent objects that are specified in the Spring configuration file. When an object is to be created, Spring uses the attribute class in the Struts configuration files to match the id attribute provided in the Spring configuration files.

There actually exist two different fashion for integrating Struts 2 together with Spring:

1. Auto-wiring Spring’s bean in Struts 2 Action classes

2. Having Spring manages the creation of Action classes

Using the first methodology the only requirement is to register a Bean like in listing 6.5, Spring will create an object of class LoginUserBoImpl and provide that object to any Action class that has a setLoginUserBO method with an argument of type LoginUserBoImpl. However, in this way the Struts 2 framework will still manage the creation of the Action class. Thus, we opted to follow the second choice: we configured the application so that Spring will create the Action class by adding a bean node to the Spring configuration file for the Action class.

Listing 6.8 shows a Struts 2 configuration file integrating Spring, while listing 6.9 shows in which way Spring takes control over objects creation.
Listing 6.8: Struts main configuration file

```xml
<struts PUBLIC

<constant name="struts.objectFactory"
  value="org.apache.struts2.spring.StrutsSpringObjectFactory" />

<action name="createUser"
  class="pt.inesc.l2f.vithea.action.CreateUpdateUserAction"
  method="createUser">...
<result name="success" type="tiles">user.list</result>
<result name="input" type="tiles">user.create</result>
</action>

</struts>
```

Listing 6.9: Spring configuration file for integration with Struts 2

```xml
<bean id="createUser"
  class="pt.inesc.l2f.vithea.action.CreateUpdateUserAction" />
```

Integrating Spring with Hibernate

Spring framework provides the integration points for permitting applications to access Hibernate services. To avoid tying application objects to hard-coded resource lookups, Spring allows to define resources such as a JDBC DataSource (the connection setup to a database from a server) or a Hibernate SessionFactory (the one delegated to the creation of Session instances) as beans in the Spring container. Application objects that need to access resources just receive references to such pre-defined instances via bean references. As discussed in section 5.2.2, for the development of the project we decided to use annotated java class instead of xml mapping file, thus the following excerpt from applicationContext.xml file shows how to set
up a Hibernate SessionFactory, which in this case is represented by the class
org.springframework.orm.hibernate3.annotation.AnnotationSessionFactory-
Bean, a subclass of Spring’s standard LocalSessionFactoryBean for Hibernate.

Also, instead of using a normal JDBC datasource we rely on c3p0, an
open source library for augmenting traditional (DriverManager-based) JDBC
drivers which robustly handles database disconnection due to session expira-
tion and reconnection in a transparent fashion.

Listing 6.10: Excerpt of Vithea’s applicationContext.xml configuration file

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"
default-lazy-init="true">
  <bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.annotation.
    AnnotationSessionFactoryBean">
    <property name="annotatedClasses">
      <list>
        <value>pt.inesc.l2f.vithea.entity.User</value>
        <value>pt.inesc.l2f.vithea.entityUserRole</value>
        <value>pt.inesc.l2f.vithea.entity.AphasiaType</value>
        .................
      </list>
    </property>

    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.show_sql">true</prop>
        <prop key="hibernate.format_sql">true</prop>
        <prop key="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</prop>
        <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
        <prop key="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prop>
        <prop key="hibernate.connection.url">jdbc:mysql://localhost/vithea</prop>
        <prop key="hibernate.connection.username">
```
Once configured the connection to the databases and the session management, these are injected to beans as shown in listing 6.11, line 3.

Listing 6.11: Dao configuration

```xml
<bean id="userDao"
  class="pt.inesc.l2f.vithea.dao.impl.UserDaoImpl">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

In this way java class UserDaoImpl will be provided by a sessionFactory instance at creation time. The next step is letting UserDaoImpl extends org.springframework.orm.hibernate3.support.HibernateDaoSupport, a convenient super class for Hibernate-based data access objects. It provides access to org.springframework.orm.hibernate3.HibernateTemplate, an helper class that simplifies data access code. Listing 6.12 provides a practical example.

Listing 6.12: Dao source code

```java
package pt.inesc.l2f.vithea.dao.impl;

import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import pt.inesc.l2f.vithea.dao.UserDao;
import pt.inesc.l2f.vithea.entity.User;

public class UserDaoImpl extends HibernateDaoSupport implements UserDao {
```
public void create(User user) {
    getHibernateTemplate().save(user);
}

public void update(User user) {
    getHibernateTemplate().saveOrUpdate(user);
}

public User get(Long userId) {
    return (User) getHibernateTemplate().
        get(User.class, userId);
}

public void delete(User user) {
    getHibernateTemplate().delete(user);
}

6.2.3 Data Tier

In our project Data Tier comprises of a database server. The Vithea system relies on MySql Community Edition [MySql 10], an open source database available under GPL license. Version used is 5.1.46-2.18.

Application’s data is stored at this layer, while is retrieved from the application layer through DAO and Hibernate, keeping it this way independent from the actual business logic. As described while introducing the Administration and Client Module on section 6.1 database maintains data related to user’s information, user exercise and user’s training session.

Table 6.1: Entity Relationship Legend

<table>
<thead>
<tr>
<th>Legend:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary key columns</td>
</tr>
<tr>
<td>Columns with constraint of foreign key</td>
</tr>
<tr>
<td>n number of related tables</td>
</tr>
</tbody>
</table>

Figure 6.7 illustrates relationships between User’s data and aphasia’s related information. Figure 6.8 shows how training exercises are structured
and existing relationships between tables. Table Directories stores information for dynamically maps file systems structure, table Document stores multimedia resources related information together with some extras information for an improved management of some functionality from the application. A unique integrity constraint has been provided to the column "title" in table Document. The remainder tables are self-explanatory.

![Entity Relationship Diagram of Vithea - User Related table](image)

Figure 6.7: Entity Relationship Diagram of Vithea - User Related table

Figure 6.9 shows how information recorded for statistics data are maintained. UserAccess table keeps track of every user’s session, storing access date, start and end time. UserExercise keeps track of the time in which a user start and end a training session. UserQuestion stores starting and ending time of each stimulus, for each attempt done.
Figure 6.8: Entity Relationship Diagram of Vithea - Exercises related table
Figure 6.9: Entity Relationship Diagram of Vithea - Tracking related table
Chapter 7

Conclusions and future work

This final chapter aims at describing an early evaluation that has been performed on the automatic recognition task (section 7.1). Besides, a summary of the goals achieved in this thesis as well as some directions for future work are described in section 7.2.

7.1 Word Naming evaluation experiments

Early experiments and evaluations have been performed on the automatic word naming recognition task. The results of these experiments have been described in the paper ”Automatic word naming recognition for aphasia treatment” submitted at the 12th Annual Conference of the International Speech Communication Association (Interspeech 2011). In the following we briefly summarize the results achieved, further details can be found in [Abad 11].

7.1.1 Data collection and Evaluation criteria

The data for these tests has been collected from therapy sessions of eight different aphasic patients. Each session comprises of naming exercises with 103 objects, each of them is shown with an interval of 15 seconds from the previous. Recordings took place in a small office with wooden walls, two inexpensive microphones were used: a head-set and a table-top microphone. Word-level transcription and segmentation were manually produced for each session. The complete evaluation corpus has a duration of approximately 1 hour and 20 minutes.

In the work carried out with the ASR a word naming exercise is considered correctly whenever the target world is spoken by the patient. However, this is not the criteria followed by clinicians in therapy sessions, where providing
the target answer after some corrections or hesitation would be considered as a failure.

7.1.2 Human vs. Automatic Evaluation

The experiments were performed using only the head-set microphone recordings. The word naming score is calculated for each speaker as the number of positive word detections divided by the total number of exercises. Analyzing the correlation between human and automatic evaluation, we find that the Person’s coefficient is 0.9043, result that we consider quite promising in terms of global evaluation. On the other side, individual evaluations of word naming recognition performance show that the system is not yet enough calibrated and that there is a strong inclination towards a high False Alarm (FA) rate. However, this is due to the telephone data used for developing the word detection approach, which were not appropriate for calibration of a word naming task with aphasia patients.

7.2 Summary and Future Work

The main goal of this thesis has been the development of a system capable of performing a necessary task: facilitating the recovery of people who have suffered a particular language disorder, aphasia.

In the last decade the number of victims of aphasia has considerably increased due to a significantly growth of one of the most common cause of this disorder, cerebral vascular accident (CVA). Fortunately, aphasia can, in most cases, be successfully recovered, but normally the patients are required to undergo via constant and frequent therapy sessions. The therapy sessions depend, of course, from the specific symptoms of the disorder, but most of these treatments aim at improving the so called patients word-finding skill. Indeed, one of the most typical residual disorder from aphasia is a problem in recalling words such as nouns and verbs. Therefore, in these cases the methodology practiced during therapies consists in Naming objects exercises. Visual or audio stimuli are shown to the patient who is asked to identify the object or activity.

In this recovery phase Vithea, the Virtual Therapist for Aphasia Treatment we have developed, will provide a valuable support by allowing therapy sessions anywhere (i.e patients’ own home, avoiding the need to reach the rehabilitation clinic), at anytime. Vithea is a multi tier Web Application designed into two different modules: Administration and Client module. The Administration module allows the management of the set of stimuli and
provides a rich Web based interface to manage the database of multimedia resources used to define new stimuli (or update existing ones). The Client module allows to perform the therapy sessions, it has been implemented by the integration of a Flash application, in order to support rich multimedia interactions (including audio recording/playing) from a standard Web browser.

The achievement of this work would have not been possible without the integration of two different, important technologies, Automatic Speech Recognition (ASR) and Web Application Framework (WAF).

ASR represents an essential building block of the system: this module is the one in charge of receiving the patient’s answers and validating the correctness of the utterances with respect to the therapeutic exercises. Various approaches have been considered for this delicate task which is particularly challenging due to the several problems associated with aphasic speech, namely hesitation, doubts and repetitions among others. Early experiments on the automatic evaluation of the recognition process have been investigated revealing promising results and suggesting future improvements.

WAFs on the other hand, constitute the backbone of the system. They have laid the foundations for a solid, modular, extensible development of the application. Specifically three mainstream open source frameworks for the development of Web Application have been integrated: Struts 2, Spring and Hibernate. Struts 2 has been used in the presentation layer, Spring has been exploited for managing the business logic of the application, while Hibernate is responsible for ensuring persistence and enhancing performance of the data layer. The system also relies on other external open source tools such as FFmpeg and a Video Encoder Engine for Adobe Flash video. These technologies also have played an important role allowing a rational and convenient management of the multimedia files associated with the designing of the training exercises.

This thesis provided me with the opportunity to deepen my knowledge in a wide range of interdisciplinary areas, ranging from medical background on aphasia disorder and its therapeutical treatment, ASR technologies, as well as mainstream open source frameworks for the development of complex Web Applications.

7.2.1 Future Work

We plan several directions as future work for the system. These entail both enhancements of the ASR approach and the integration of new sophisticated tools. The ASR can be improved by providing a better calibration of the
system operation point to the specific user profile. Also, the chance to incorporate tools like goodness of pronunciation [Witt 99] will be investigated, as it could allow a different type of the assessment of the pronunciation errors.

Concentrating on the features offered by the system, we will certainly integrate the possibility to provide help to the user whenever it will be needed. That is, the "virtual therapist" will be able to supply both semantic and phonological hints to the patient during the therapy sessions. Different forms of help will be analyzed, varying from a set of alternative words in which only one is right, to a hint about the first letter of the targeted word. Help could be given both in the form of a written solution or as a speech synthesized production based on Text to Speech (TTS). Actually, TTS is an enhancement that will be integrated into the system in the short term.

Moreover, we plan to incorporate an intelligent animated agent who will exploit the support for synthesized speech to behave like a sensitive and effective clinician, providing positive encouragements to the user. In this way we hope that therapy sessions will result more natural and comfortable making the system a relaxing means for recovering aphasia.

In a long term vision we also plan to perform a comparative evaluation of the medical benefits enjoyed by patients who used the system versus patients who followed ordinary therapy sessions.

Finally, in a complementary way to this enhancements, we are already evaluating to use and extend the system for the treatment of other forms of speech disorders like for instance, Dysarthria, and for Dementia.
Bibliography


International Congress of Phonetic Sciences (ICPhS), Barcelona, Spain, 2003.


